IDA

INSTITUTE FOR DEFENSE ANALYSES

Broker Performance

Norman R. Howes
Edward A. Feustel

May 1998

IDA Paper P-3439
Log: H 98-002928

This work was conducted under contracts DASW01 94 C 0054/ DASW01 97 C 0056, Task A-209, for the Defense Advanced Research Projects Agency. The publication of this IDA document does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that Agency.

INSTITUTE FOR DEFENSE ANALYSES

Broker Performance

Norman R. Howes
Edward A. Feustel

PREFACE

This document was prepared by the Institute for Defense Analyses (IDA) under the task order, Advanced Information Technology Services Architecture. It provides an assessment of particular broker technologies relevant to that architecture. The work was performed for the Information Systems Office, Defense Advanced Research Projects Agency (DARPA).

This document was reviewed by Dr. Dennis Fife, Dr. Reginald Meeson, Dr. Richard Morton, Mr. David Wheeler, and Dr. Glen White, all of IDA, plus Dr. Craig Thompson of Object Services and Consulting, Inc.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

By their very nature, command and control (C2) systems are typically distributed. Timely and efficient data transfer among the distributed nodes of such systems is thus critical. In recent years, software known as brokers has gained increasing attention for effecting data transfer among the components of distributed information systems. Brokers facilitate system design and operation by mediating the interaction between processes executing on separated nodes so that the processes do not need detailed information about one another (e.g., physical location or network address). Within the DARPA C2 community, most attention has been given to object request brokers (ORBs), implemented according to the Common Object Request Broker Architecture (CORBA) specification. However, another class of brokers, known as subscription brokers, is also of interest.

C2 systems can have stressing timeliness demands, for example, requiring the transfer of significant quantities of data in seconds of time. Even when timeliness demands are not so extreme, efficient data transfer is still necessary so that valuable communications bandwidth is not being withheld from other applications. The question thus arises as to how effective brokers are in satisfying the data transfer needs of C2 systems. To help address that question, this paper seeks to capture what is known about broker performance and planned enhancements and to synthesize this information in one place.

Current CORBA Performance. While there exists much discussion about CORBA ORB performance, a review of the literature in fact revealed very little quantitative data available from controlled experiments. One limited set of experiments noted in the main body of this paper suggests that the performance of current CORBA ORB implementations may not be adequate in demanding situations. The experiments indicated that ~200 air vehicle tracks could be moved per second between distributed nodes. One might expect an air picture to be far more dense than this, especially when false images from electronic countermeasures are included and the tracks of the same object generated by multiple sensors are taken into account. However, the results of the stated measurements are by no means definitive. On one hand, more effective designs than used in the experiments are possible (e.g., grouping of tracks), while on the other hand the experiments were highly idealized laboratory ones with no competing demands on the system components from processes other than data transfer. Further tests are certainly called for.

Existing data does not appear adequate to ascertain if CORBA ORBs meet the needs of C2 programs such as those of DARPA.

Subscription Brokers. CORBA ORBs are built around the request-reply concept – i.e., a client requesting execution of a process in a server through the broker. In this way, whenever a client needs data from the server, the client requests that the data be transferred. Subscription brokers are based on a different, so-called subscribe-and-publish model. There, the client places a subscription with the server so that whenever the server updates the data, it automatically sends the new data to the client.[1] Subscription brokers thus have the inherent advantage of prepositioning necessary data with the client (and the filtering mechanisms in subscription brokers can be used to avoid forwarding unnecessary data). Thus, execution times for overall processes involving remotely maintained data would be faster with subscription brokers.

Apart from the advantages in prepositioning data, subscription brokers also appear to be faster than ORBs in transferring data (the reasons being limitations imposed by CORBA's reliance on the request-reply paradigm). Limited experiments noted in the main body of the paper showed that subscription brokers could be one to two orders of magnitude faster. However, as with the experiments noted above, these results can only be regarded as exemplary and not definitive.

Future CORBA Performance. Several enhancements for CORBA performance directly relevant to C2 systems are currently being planned. Information on these enhancements was pulled together from a number of sources and given in the main body of this paper. Highlights are summarized here:[2]

- Notification Service. Provides full subscribe-and-publish capability. While this enhancement might not fully match the performance of the subscription brokers noted above due to reasons of underlying CORBA implementation requirements, it will give CORBA the full capability of prepositioning data noted for those brokers. Furthermore, the capability will be provided in a standardized way (current subscription brokers have vendor-specific interfaces).
- Data Stream Control. Allows for data streaming not under the control of the CORBA ORB. Performance enhancements precluded by CORBA requirements (e.g.,

[1] CORBA has limited such capability today in terms of its Event Service, but it is lacking, e.g., in terms of message processing efficiency and the ability to filter the data that is sent.

[2] Other services discussed in the main body of the paper are the Asynchronous Messaging Service and the Multiple Interface and Composition Service.

multicasting) can thus be realized. While data stream control is primarily being proposed for use with audio/visual data, it can be used for any type of data that is approximately continuous in its generation (e.g., air vehicle tracks).

- Mobile Agent Facility. Provides the option of sending a program to the data, rather than requiring transfer of the data, when that is the more efficient approach.
- Real-Time Service. Allows greater control of system resources, thereby improving the ability of applications to achieve predictable performance. However, predictability by itself does not necessarily guarantee that desired timeliness goals will be met.
- Objects by Value. Allows methods to return the value of an object, in addition to the current capability of returning a reference to the object, thereby substantially improving the efficiency of transferring data objects from one host to another.

Estimates of availability vary by service type, but generally implementations of these services are expected in the period from the last quarter of 1998 to the second half of 1999. Of the services listed, the Notification and Data Stream Control Services appear the most significant. They offer the potential for significantly enhancing data transfer performance in C2 systems.

Recommendations for DARPA Action. This paper is intended primarily for the DARPA community involved with C2 applications, which fall primarily under the DARPA Information Systems Office (ISO), to include the DARPA-DISA Joint Program Office. Implications for DARPA action follow from the material presented here. In particular:

- *That DARPA ISO form a performance testing program to address data transfer effectiveness in C2 applications more explicitly.* As noted, quantitative performance measurements are very limited. Furthermore, DARPA C2 programs do not appear as yet to be paying major attention to performance issues. But without attention to the details of data transfer performance, the goals of the C2 programs may not be met. The activities of a performance testing program to support these goals are given in the main body of the paper.
- *That the DARPA ISO community become more active in tracking and influencing the actions of the Object Management Group (OMG) relating to CORBA data transfer performance.* Currently, the DARPA community has only very limited interaction with the CORBA community. But, as evidenced in the planned services listed above, there is much activity and interest in the CORBA community (which is coordinated under the OMG) in providing enhanced ORB performance in ways directly relating to

C2 applications. Interaction between the two communities should be mutually beneficial.

- *That the DARPA ISO C2 programs begin addressing how their applications will need to be modified to use the new CORBA data transfer capabilities*, if they intend to continue using CORBA. The planned CORBA services noted above could significantly enhance C2 application performance. Yet the applications currently being developed in the ISO C2 programs are most likely designed for existing CORBA capabilities. Significant redesign could be required, for example, to make use of the subscribe-and-publish capabilities that will be provided.

To coordinate the activities involved in each of these recommendations, it is suggested that DARPA ISO form a performance working group from the development teams for the relevant ISO projects. This working group would give impetus to the recommended activities and help share their results across the ISO programs.

1. INTRODUCTION

1.1 Objective and Scope

The timely and efficient transfer of data among distributed nodes is obviously critical to the functioning of command and control (C2) systems. In some instances (e.g., air engagement control), significant amounts of data (e.g., aircraft tracks) must be moved in the order of seconds of time. Even when the demands of timeliness are not so extreme, efficient data transfer is still required so that valuable communications bandwidth is not withheld from other applications.

Modern information systems are being developed using the object-oriented paradigm because of the benefits ascribed to this approach, e.g., facilitating the reuse of software components and the interoperability among system nodes. Object Request Brokers (ORBs) are a general means for effecting the interaction among distributed nodes and, in particular, for transferring data. There are other means for organizing distributed systems (e.g., remote procedure calls, agents), but this paper concentrates on brokered architectures, and in particular on the question of how well the ORBs support the timely and efficient transfer of data.

These ORBs are receiving significant attention for use in C2 systems, e.g., in the DARPA Advanced Information Technology Services (AITS) program. Current thinking is mostly on the use of ORBs in the interaction between upper level command posts, but the ultimate intent is to apply them at the tactical level, too.

The Common Object Request Broker Architecture (CORBA) is a specification for implementing ORBs. It is accompanied by a number of other specifications for supporting CORBA services. These specifications are developed as standards by the Object Management Group (OMG). In fact, CORBA is the only source of ORB specifications receiving wide recognition. Thus, CORBA ORBs are the main focus of this paper. There is, however, a different class of brokers called Subscription Brokers (explicitly defined in Chapter 3) that will also be addressed since they offer an alternative to some of CORBA's currently realized capabilities.

The objective of this paper is to characterize what is known about current CORBA performance in transferring data and to identify what enhancements may be expected, particularly as this performance in transferring data might relate to distributed C2 systems. The paper does not develop specific requirements for C2 systems and relate them to detailed measurements of CORBA performance. Indeed, the range of functionality considered for C2 systems is far too broad for this short paper to attempt to identify or characterize the consequent requirements. Furthermore, the experimental base of results available on CORBA performance is very limited.

Rather, this paper has the more modest goal of capturing what is known about CORBA performance and planned enhancements and synthesizing this information in one place. This would seem valuable to do because such a compendium of information does not now exist. It should be useful to those (e.g., program managers) who need to make decisions on CORBA or related technology implementations for C2 systems. In that regard, this paper is particularly intended for individuals in DARPA and the related community who make such decisions.

1.2 Command and Control Context

As noted in the previous section, this paper does not present command and control requirements. However, for the sake of context, it is still necessary to provide some general concept of command and control capabilities, in particular envisioned future capabilities since the applications using ORBs are looking toward next generation (and beyond) C2 systems. The report of the Task Force [ABIS TFR] that developed the concept for the Advanced Battlespace Information System (ABIS) provides that context.

Battle management may be regarded as the overall function that the composite of C2 systems supports. The ABIS report describes battle management as:

"Battle Management includes operations planning, force management and coordination, and direction of C3ISR during mission execution. It spans current operations through future operations to future plans, with the corresponding situation/crisis assessment and operational evaluation at each level. Battle management deals with multiple decision loops including a fast, sensor-to-shooter decision loop... It also deals with complex issues like the uncertainty of large amounts of information and aggregation of many variables into the assessment of progress in achieving a given objective."

Furthermore, regarding the sensor-to-shooter coupling the report goes on to state:

"Effectively executing combat operations in a joint force environment involving many ground, air, space, and shipboard resources entails two key challenges:

1. From within a universe of many joint force resources, individual sensors and shooters must be tasked and provided with the necessary priorities and targeting information needed to carry out multiple specific missions against multiple specific targets to achieve all of the battle manager's objectives. The development and maturation of processors to assist in decision making and optimization of finite sensor and weapons assets is the first of two key challenges. This challenge is referred to... as coordination of missions.

2. For each individual mission, the information linkages must be established between sensors and shooters to enable the timely execution of missions, especially time-critical missions. Because,

ideally, the sensors can be time shared among many shooters (in addition to the battle manager), effective and efficient implementation of these linkages and the ability to pass information through them will inevitably require the establishment of execution controllers performing real-time or near real-time C2 operations...

In this environment, the key operational concept required is one of distributed command and control, with an execution controller for each sensor-to-shooter execution team (which is really a sensor-to-C2-to-shooter team) performing many of the same functions that the battle manager performs. However, the sensor-to-shooter team plans how the mission is to be executed, whereas the battle manager plans what will be executed. Thus, the C2 for each sensor-to-shooter team requires functional capability similar to that of the battle manager, but for an increased depth of detail spanning a reduced breadth of area of interest and having a much stronger focus on the timeliness of the information versus its completeness."

The following figure from the report nicely displays the overall functions involved in battle management and the nesting of sensor-to-shooter operations within battle management. Furthermore, the figure roughly notes the time scales involved – which vary from seconds to hours. The shorter time scale might be the most pressing from the standpoint of data transfer, but the longer intervals could also be demanding if very large quantities of data are involved.

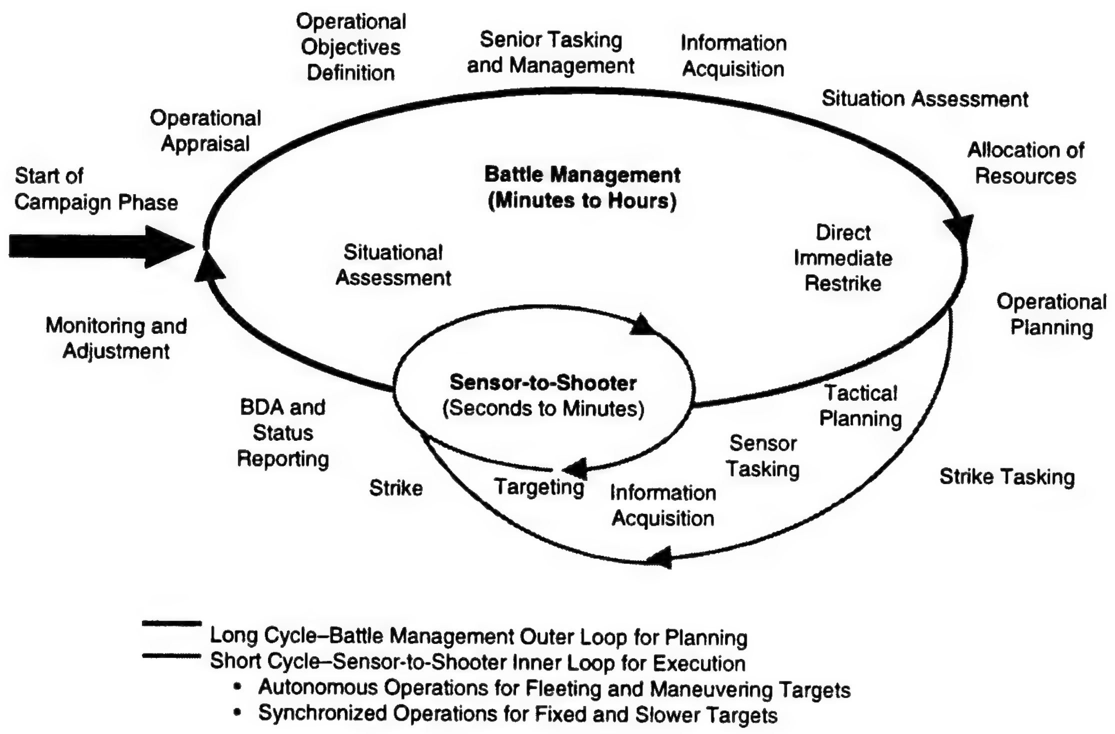

Operational Objectives Definition
Senior Tasking and Management
Information Acquisition
Situation Assessment
Operational Appraisal
Allocation of Resources
Start of Campaign Phase
Battle Management (Minutes to Hours)
Direct Immediate Restrike
Situational Assessment
Operational Planning
Monitoring and Adjustment
Sensor-to-Shooter (Seconds to Minutes)
Tactical Planning
BDA and Status Reporting
Sensor Tasking
Strike Tasking
Strike
Targeting
Information Acquisition
Long Cycle–Battle Management Outer Loop for Planning
Short Cycle–Sensor-to-Shooter Inner Loop for Execution
• Autonomous Operations for Fleeting and Maneuvering Targets
• Synchronized Operations for Fixed and Slower Targets

Figure 1. Operational Concept for Integrated, Target-Focused Operations

Conceivably, one could analyze the functions involved in battle management to characterize the data transfers involved in terms of the quantities and timeliness required. That is not within the scope of this paper, but a simple illustrative example might be useful just to get an order-of-magnitude sense. For air engagement control, the ABIS concepts envision a "zone many aircraft tracks would be picked up by the sensors and have to be exchanged. For the sake of discussion, it does not seem unreasonable to assume that 100 aircraft and missiles, enemy and friendly, could be involved in the battlespace. However, electronic countermeasures could produce multiple or false images, thereby multiplying the figure severalfold. Furthermore, detection of an aircraft by multiple sensors could also increase the number of tracks (prior to track correlation) that had to be circulated. Thus, it is not at all unreasonable to think that there might be at least 1000 tracks that would have to be circulated. The update rate for this information is not certain, but it would seem to be of the order of a second or seconds, especially when weapon commitment becomes imminent.

1.3 Organization of Paper

Chapter 2, which follows next, presents the information that could be gathered on the performance of current CORBA implementations. Since subscription brokers offer an alternative to some of the current CORBA capabilities, their concept and some performance data are given in Chapter 3. Significant future enhancements are planned for CORBA, so that is the subject of Chapter 4. Then, based on the material presented in these three chapters, Chapter 5 discusses actions DARPA might take to be most effective in incorporating ORBs and related technologies in their C2-related research and development programs.

2. CURRENT CORBA PERFORMANCE

This chapter presents information about the performance of current CORBA implementations. While there are several articles in the literature about CORBA performance, very little in the way of documented quantitative experimental data could be found. Recently, with the attention of the CORBA community focusing more on matters of performance, performance results are being cited on mail lists and on the web, but the environment in which the experiments were conducted is rarely described in any detail. This chapter is thus confined to two examples where sufficient quantitative data could be found.[3]

2.1 Results of IDA Experiments

Several experiments have been performed at IDA to understand basic timing relationships between CORBA objects on certain platforms of interest. These performance tests do not take into consideration the overhead introduced when multiple clients and servers compete with each other simultaneously for system resources. Thus, the performance demonstrated in the tests may be above that attainable with real-world systems designs. In other words, these tests serve to identify upper bounds in system performance for the given operation considered.

2.1.1 Experiment No. 1

Measurements were made with the Orbix 2.1 CORBA implementation. They are documented in [IDA P-3327]. They include (1) determining the number of remote method invocations per second that can be achieved on distributed objects residing on various platforms and (2) the number of data objects that can be transferred per second between objects residing either on the same platform or on pairs of different (not necessarily homogeneous) platforms.

Four cases were considered:

- Case 1.1: Client and server objects on the same Sparc20.
- Case 1.2: Client and server objects on different Sparc20s.

[3] Another possible source of performance measurements is the Joint Task Force Advanced Technology Demonstration program, but it was not possible to obtain any data from that source.

- Case 1.3: Server object on Sparc20, client object on P90 NT.
- Case 1.4: Server object on Sparc20, client object on Sparc IPX.

All unix platforms used the Solaris 2.5 operating system,[4] and in cases 1.2-1.4, a 10 Mbps Ethernet network was used to connect the platforms. In all cases there were no other users on the platforms and the network experienced little other traffic (as verified by use of netstat). The entire computational resources were being used to make the remote method invocation or data transfers.

Three quantities were measured:

- The number of remote method invocations that could be made per second.
- The number of byte strings of length 938 bytes (the length of a GCCS track object) that could be moved from a server object to a client object per second.
- The number of actual GCCS track objects that could be moved from a server object to a client object per second.

In the case of the string or track object transfers, a separate client request was used to transfer each string or object, respectively. A sample of 10,000 measurements was made for each of these quantities and for each of the four cases (Cases 1.1–1.4). Thus, in addition to giving the measurement of the quantity in average number/second, it was also possible to indicate the slowest and fastest measured quantities for the 10,000 measurement, and the standard deviations associated with these measurements. The results of these measurements are given in Table 1.[5]

These results may be related to the air engagement control example noted in Section 1.2. The cases that are probably the most relevant for this example are the ones where the client and server objects are on separate platforms. In these cases, the numbers of tracks that can be transferred per second are in the 150 to 272 range. Such figures are significantly below the notional requirements estimate of 1000 tracks per second given in Section 1.2. Furthermore, the results of these experiments are for idealized circumstances where all system resources were devoted to track data movement and not to any of the many other functions that would be necessary in air engagement control. Thus, the 150-272 figure could be significantly less in a real-world application. On the other hand, given such results, different design approaches (e.g., transferring groups of track together) might be taken to improve performance. Thus, the results

[4] If the experiments were repeated with the more recent Solaris 2.6, some increased performance would be expected since the implementation of TCP/IP has been improved relative to version 2.5.

[5] Incidentally, by way of comparison for the remote method invocation figure, an ordinary procedure call on a Sparc20 takes 1.5 *microseconds* (3 orders of magnitude less).

here cannot in any sense be regarded as definitive, but they do suggest there can be cases where current CORBA implementations will not provide adequate data transfer capability.

Table 1. Results of IDA Experiment No. 1

Platform	Measured Quantity	Slowest (msecs.)	Fastest (msecs.)	Average (msecs.)	Std. Dev. (msecs.)
Case 1.1: Client & server objects on same Sparc20.	Remote method invocations: 588/sec. (avg.)	25	1.62	1.7	0.37
	Byte strings of length 938 bytes: 538/sec.	19.5	1.8	1.86	0.32
	GCCS track objects: 385/sec.	19.7	2.5	2.58	0.32
Case 1.2: Client & server objects on different Sparc20s.	Remote method invocations: 606/sec. (avg.)	135	1.59	1.65	1.44
	Byte strings of length 938 bytes: 373/sec.	23.8	2.63	2.68	0.43
	GCCS track objects: 272/sec.	19	3.6	3.67	0.32
Case 1.3: Server object on Sparc20, client object on P90 NT.	Remote method invocations: 246/sec. (avg.)	43.5	3.6	4.06	0.55
	Byte strings of length 938 bytes: 196/sec.	48.6	4.7	5.1	0.62
	GCCS track objects: 150/sec.	45.2	6.2	6.65	0.59
Case 1.4: Server object on Sparc20, client object on Sparc IPX.	Remote method invocations: 320/sec. (avg.)	58.2	3	3.12	0.97
	Byte strings of length 938 bytes: 225/sec.	65.8	4.27	4.45	1.04
	GCCS track objects: 166/sec.	57.1	5.8	6.04	0.41

It can also be seen from these results that a track server that has to serve up a large number of tracks per second to multiple clients would provide these clients with a fairly unstable flow of track data. This is because at least once every few seconds a client would experience one of the approximately 20 millisecond outliers. For real-time applications, one prefers standard timing deviations that are at least three orders of magnitude smaller than the average and preferably much smaller. This is so the software does not have to deal with abnormal conditions every second.

2.1.2 Experiment No. 2

After the first set of experiments was completed, a companion set was undertaken to determine how the results of some of the previous experiments might change if Unix sockets were used directly for data transfer rather than CORBA. The reason for these experiments is that, in involving direct data transfer without any ORB involvement, they serve to illustrate what component of the transfer times is due to the network software (e.g., execution of protocols). These experiments were limited in nature and not as extensive as those documented in [IDA P-3327]. The experiments involved two Sparc20 platforms with an Ethernet connection. TCP sockets were used, since TCP underlies the CORBA data transfer.

Since there is no concept of remote method invocations using sockets, and since there is no socket support for translating objects between heterogeneous platforms, the only tests that were performed involved transmitting a 938 byte string (the length of a GCCS track) to the client from the server after the client requested it. With this technique, object transfer will work between homogeneous machines. The usual way of doing this is to use the Unix utility "bcopy" to copy the object into a string before transmitting and then using it again at the other end to copy the string back into an object.

With the client object and the server object on different Sparc20 platforms, ten samples of 10,000 tracks each showed that 1,332 tracks could be transferred per second (on average). No minimums, maximums or standard deviations were recorded. This is roughly a four-fold increase in tracks over the analogous CORBA case (Case 1.2 above). Or, in other words, the network software accounts for about one-quarter of the overall CORBA data transfer time for the configuration considered.

2.1.3 Experiment No. 3

The experiments noted above can quickly become dated because of the speed at which platform processing capability is increasing. To gain some insight on the changes, the CORBA tests reported above (Experiment No. 1) were repeated on a more powerful (single-processor) UltraSparc platform. In addition, to gain some insight into the differences among CORBA implementations, three different products were also considered – Visibroker 3.0, Orbix 2.1, and PowerBroker. In all of the experiments, the client and server objects were on the same platform.

For the case where remote invocations were counted, the Visibroker implementation achieved 1707 per second, Orbix 1158 per second, and PowerBroker 638 per second. For the case where transfers of byte strings of length 938 bytes were counted, the Visibroker implementation achieved 1550 per second, Orbix 1111 per second, and PowerBroker 599 per second. For the case where transfers of track objects were counted, the Visibroker implemenation achieved 1123 per second, Orbix 790 per second, and PowerBroker 363 per second.[6]

The effect of the more powerful processor (UltraSparc vs. Sparc20) is clear when these results are compared to Experiment No. 1 above. The results for the Orbix implementation (the ORB used in Experiment No. 1) improve by approximately a factor of two. What is perhaps even more interesting in the results is the differences shown between ORB implementations. The Visibroker results are approximately a factor of three better than those for PowerBroker.

2.2 Washington University Results

Web searches led to a number of references to papers in the current literature. Most of these references are by researchers at or connected with two universities. The most prolific is a group of researchers at the Washington University in Saint Louis. Also well represented is a group of researchers at the University of Rhode Island and their colleagues. Together, these two groups probably account for over two-thirds of the papers identified on CORBA performance. However, only the Washington University papers contain quantitative experimental results. While the University of Rhode Island papers contain no published results, they do state the need for improving CORBA's real-time performance based on their prototyping experience.

The publication from the Washington University research group that seems most relevant is [WUSL 96a]. This is a paper presented at the ACM SIGCOMM Conference in August, 1996 by A. Gokhale and D. Schmidt with the title of Measuring the Performance of Communication Middleware on High-Speed Networks. While much of this group's research is dedicated to the study of transferring large byte-strings of video data over ATM networks, such as in [WUSL 96b], this particular paper considers the transfer of a variety of data items of basic types (integers, characters, double precision numbers, and simple 32 byte structs).

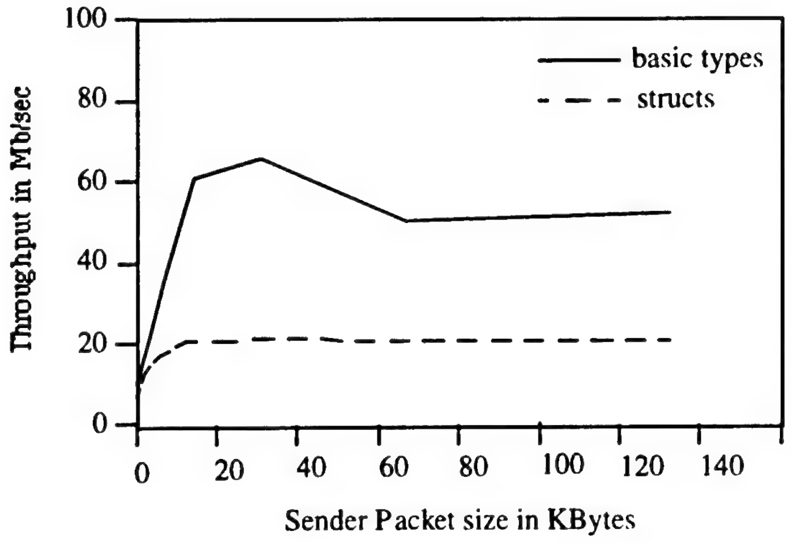
These experiments differ in three significant regards from those in Section 2.1: much larger data packets are generally transferred, the packets are sent immediately following one another (rather than having the client request each packet individually), and a faster network is used. Each of these factors is expected to contribute to faster data transfer results. While the mode of operation in these experiments does not seem applicable, for example, to cases such as

[6] Similarly, there was an accompanying experiment using Unix sockets. In this case, in ten samples of 10,000 byte strings (of length 938), an average of 13,448 transfers per second was achieved.

the air tracks considered in Section 2.1, the experiments do illustrate other aspects of CORBA data transfer. The results of the Washington University experiments may be interesting for applications that need to stream large frames over a network like video or teleconferencing applications or satellite downlink streams.

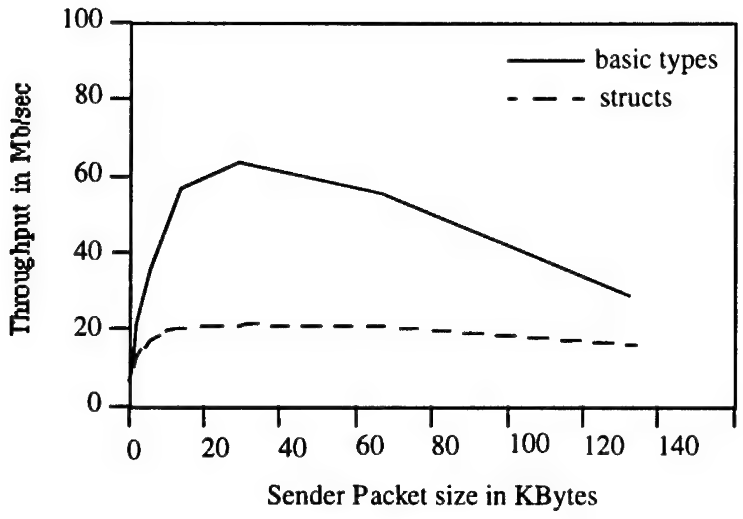
In the experiments described in [WUSL 96a], the basic data items were packaged in large packets of 2^nK bytes (1K = 1024 bytes) for n = 0 . . . 7. Thus, the packet sizes considered were between 1K = 1,024 bytes and 128K = 131,072 bytes. The results consisted of plotting the throughput in Mbps against packet size for two different CORBA implementations, for two different socket mechanisms, and for an RPC mechanism. These experiments used a 155 Mbps ATM network, the SunOS version 5.4 operating system, and a pair of dual processor Sparc20s. The two CORBA implementations tested were the Orbix and ORBeline ORBs.

The results of the Washington University experiments for the two CORBA ORBS are summarized in Figure 2 and Figure 3. The graphs in the paper show more detail. For instance, there are separate curves for all the basic data item types tested, but since the results were almost the same for all types, they are all lumped into a single curve here. The measurements in Section 2.1 can be related to the ones given here by noting, for example (Case 1.2, Section 2.1), that 373 byte strings per second of length 938 bytes is equivalent to 2.8 Mbps. Thus, for the parameter domain explored in the Washington University experiments, the data transfer rates are about a factor of ten greater.



Source: WUSL96a

Figure 2. Performance of the Orbix ORB



Source: WUSL96a

Figure 3. Performance of the ORBeline ORB

As the size of the objects grows exponentially, as they do in these experiments, one might expect the throughput plotted against packet size to also grow significantly and then perhaps flatten. This is because as the packets get bigger, more data is moved per ORB operation, so the CORBA overhead can be expected to decrease until it eventually becomes insignificant. However, such behavior is not borne out in the results of the experiments. According to the observations as shown in the figures for both versions of CORBA, there appears to be an optimal packet size of 32K bytes. CORBA overhead decreased from 1K packet sizes up to 32K packet sizes. Thereafter it begins increasing again. This is especially notable with the ORBeline ORB where performance decreases to about 40% of peak when the packet size is 128K bytes. The Orbix ORB is not as marked, decreasing to only about 75% of its peak performance when the packet size reaches 128K bytes. This phenomenon probably has as much to do with the size of buffers used as with the specific CORBA implementation.

Another interesting phenomenon is that this fall-off in throughput after this optimum packet size is reached does not seem to occur when transferring objects as opposed to byte-strings. In the case of objects, CORBA throughput is reported to be only about one-third of that achievable with byte-strings, but when it reaches the maximum at about 32K byte packet sizes, it levels off and stays there.

The Washington University researchers also did some loop-back experiments with the SparcStation loopback device to get a rough idea of how the ORBs might perform on even higher speed networks (ideal unlimited bandwidth networks). The results for moving structs (objects) were pretty much unchanged. The results for moving the basic types encapsulated in large packets increased to just over 100 Mbps for the Orbix ORB and almost 200 Mbps for the ORBeline ORB. Moreover, there was no peak at 32K packet sizes. In this case the Orbix ORB reached its maximum at the 64K-packet size and then remained constant while the ORBeline ORB appeared to approach its limit asymptotically.

The researchers compared using sockets with both a 155 Mbps ATM network and with the SparcStation loopback device. For the ATM network they achieved about one-third better throughput and the peak was at 16K byte packet sizes with the fall-off not being as severe as in the CORBA case. In the socket loop-back experiments, the graphs reached nearly 200 Mbps at a packet size of 32K bytes and essentially stayed there as the packet size increased.

3. SUBSCRIPTION BROKERS

Subscription brokers provide a means for transferring data from servers to clients. These brokers may be regarded as an alternative to current object request brokers for some uses, or as a possible new capability to be realized in future object request brokers. Accordingly, this chapter explains the concept and use of subscription brokers, and gives rough performance comparisons between currently existing subscription brokers and CORBA implementations.

3.1 Basic Concept

The term "subscription broker" refers to a class of middleware products that implement a subscribe-and-publish paradigm for interfacing client and server objects in a distributed system. This is in contrast to the request-reply paradigm of middleware products like the Distributed Computing Environment (DCE) and CORBA. The distribution model assumed by DCE and CORBA is that of the remote procedure call (or the remote method invocation). DCE does not include the concept of a broker but CORBA does. In the case of CORBA, the object request broker(s) is used to locate remote objects and perform a variety of services on behalf of the client objects that makes dealing with remote objects easier.

With the subscribe-and-publish paradigm, the distribution model is one of subscribers and publishers. Rather than calling a procedure in a remote server or invoking a method in a remote object, the client subscribes to one or more named categories of data (or objects) that a server maintains. Whenever updates are made to those categories of data, the server publishes these updates to all of the subscribing clients. All of the subscribing clients continue to get all updates until such time as they cancel their subscription. The request-reply model is a data-pull model whereas the subscribe-and-publish model is a data-push model.

The subscribe-and-publish model relieves the client objects from the necessity of having to make a request every time they need data. Instead they request it once and continue to get updates to the data for as long as they maintain their subscription. The subscribe-and-publish model also relieves the server object of having to send reply messages in response to all the request messages. Instead, the server either broadcasts or multicasts the message once and all of the subscribers receive it.

Not all subscribe-and-publish middleware products provide brokers, just like all request-reply middleware products do not provide brokers. But a subscription broker can provide many advantages for the clients in a subscribe-and-publish style distributed system just as an object request broker can provide many advantages for the clients in a request-reply style distributed system. Subscription brokers, like object request brokers, can provide object related services to their clients, but many do not provide as rich an object manipulation environment as object request brokers. However, many CORBA middleware vendors either offer subscribe-and-publish products that are not CORBA compliant, but which provide most of the CORBA object management services, or have announced they will do so in the future. For example, Iona, the producer of the Orbix version of CORBA, has had a subscribe-and-publish product called OrbixTalk available for some time now.

3.2 Uses of Subscription Brokers

The particular functionality subscription brokers offer is the ability to preposition data with applications (clients) and keep that data updated as it changes at its external sources. This prepositioning will lead to shorter execution times for processes needing that data, as compared to having to fetch that data from the remote sources when needed. This can be particularly important for systems where timeliness of execution is important, e.g., distributed command and control systems. However, one does not want such a data-push scheme where data is transferred that the client does not need or the data is transferred at an update rate much greater than the client needs,[7] since this would needlessly use up computation and communication resources. Subscription brokers have the capability to set update rate and determine the particular data elements to be updated, including the ability to do this in a dynamic manner based on changed subscription requests from the clients.

The ability to preposition data is available in CORBA through the existing Event Service. While this is of some utility, the Event Service only provides limited specificity over what data to send and the Event Service does not provide the full capability and performance available in subscription brokers, as discussed below in Section 4.2. The new Notification Service proposed for CORBA should provide greater capability in this regard, as also discussed in Section 4.2.

One could use CORBA to repeatedly poll the data source to get updates to build the prepositioned database. But that could lead to excessive calls since the client application does

[7] For example, if there are repeated incremental changes to the data and the difference in individual increments of change is not important to the application.

not know when the data changes and hence when to poll, thereby leading to excess usage of computation and communication resources. Even if one could determine a poll rate that avoided null responses, use of a subscription broker could still be preferable because it can effect the data transfers faster. As shown below in Section 3.4, current subscription brokers can be many times faster than existing CORBA implementations in transferring data.

In summary, subscription brokers offer significant capabilities not present in existing CORBA implementations. At the same time, though, CORBA offers a wider range of general purpose object services than do existing subscription brokers. Thus, applications such as a command and control system could well require the capabilities of both subscription and object request brokers.

3.3 Current Commercial Activity

At the present time there is significant activity in the data-push middleware market. At least three dozen different products are on the market, with perhaps half of them providing subscription brokers (see Table 2, taken from [CUS97]). There is at least one standardization initiative underway with over forty member organizations including Sun Microsystems, Cisco Systems, Informix, JavaSoft, NETCOM, InfoSeek, CyberCash and TIBCO. They are proposing an object-oriented version of the Information Bus concept (which they call InfoBus) and circulating a draft specification for public review to be used as the basis for interconnecting Java Beans and Applets into Web applications. Recently, the OMG sought to bring this initiative under the CORBA umbrella. In response, the major players in this initiative are cooperating with the OMG in the context of the proposed Notification Service (see Section 4.2).

The original Information Bus concept came from TIBCO. They market it under the name of TIBNet (The Information Bus Network). TIBCO has successfully marketed its middleware approach to over 400 banks and brokerage houses that use TIBNet to exchange real-time stock and currency fluctuation quotations worldwide. TIBNet is being put forward as the putative standard for the CORBA Notification Service.

3.4 Subscription Broker Performance

At present, little quantitative information is available about the performance of subscribe-and-publish middleware products, although they are claimed to be more efficient than middleware products based on the request-reply paradigm. This might be expected because request-reply semantics are not required and individual requests are not used for each data packet sent (which was the mode of operation for the CORBA experiments in Section 2.1).

Table 2. Data-Push Middleware Products

Company	Product	Client	Server	Publish/ Subscribe	Web Push Technology	Push Transactional
Active Software Mountain View, CA	ActiveWeb Information Broker	Windows 3.1, NT, & 95	Unix, Windows NT	X		
BackWeb Technologies San Jose, CA	BackWeb	Windows 3.1, NT, & 95			X	
BEA Systems Inc. Sunnyvale, CA	BEA Tuxedo	Windows NT, Unix	Windows NT; Unix	X		X
Dazel Corp. Austin, TX	MetaWeb	Browser on any client	HP-UX; AIX; Solaris	X		
Forte Software Inc. Oakland, CA	Forte	Windows 3.1, NT & 95 Motif; Macintosh; Unix	Window NT; Unix; Solaris; VMS			X
InCommon San Mateo, CA	Downtown	Windows NT & 95	Wiindows NT; Unix		X	
InterMind Corp. Seattle, WA	Communicator; Dynamic Publisher	Windows NT & 95; Solaris	Windows NT; Solaris		X	
Lanacom Inc. Toronto, Ontario, Canada	Headliner Professional	Windows NT & 95	N/A		X	
Level 8 Systems New York City, NY	L8MQ/Event; L8MQ/Message Broker	Windows NT: Java/Internet	MVS CICS; AIX; Windows NT; Solaris	X		X
Marimba Inc. Palo Alto, CA	Castanet	Windows NT & 95	Windows NT & 95; Solaris		X	
Momentum Software Corp. Hasbrouck Heights, NJ	XIPC	Wiindows 3.1, NT & 95; Macintosh; OS/2	VMS; Unix; MVS	X		
Neon Inc. Englewood, CO	NEONet	Windows NT & 95	Solaris; Windows NT	X		
Netweave Corp. Philadelphia, PA	Netweave	Windows 3.1, NT & 95; OS/2; Macintosh	MVS; VMS; Unix; Windows NT	X		
OpenHorizon Inc. San Francisco, CA	Ambrosia	Windows 3.1, NT & 95	Windows NT	X		

Table 2. Data-Push Middleware Products (continued)

Company	Product	Client	Server	Publish Subscribe	Web Push Technology	Push Transactional
Passport Corp. Paramus, NJ	IntRprise	Java clients	Internet/intranet servers			X
PointCast Cupertino, CA	PointCast	Windows 3.1, NT & 95	Windows NT		X	
Precise Software Solutions Braintree, MA	Q/Booster	Unix; DOS; OS/2	MVS; VM; Unix	X		
Prolifics, a JYACC Company New York City, NY	Prolifics 3	Windows 3.1, NT & 95; Unix; SunOS; Solaris	Windows NT; Unix; Solaris			X
StarBurst Communications Corp. Concord, MA	Starburst Multicast	Windows NT & 95; Unix, OS/2	Wiindows NT; Unix		X	
Talarian Corp. Mountain View, CA	SmartSockets	Windows 3.1, NT & 95; OS/2	Windows NT; Unix; Open VMS	X		
Tibco Inc. Palo Alto, CA	TIB Message Bus; Rendez-vous; TIBnet	DOS, Windows 3.1, NT & 95	Windows NT; VMS; Unix	X		
Transarc Pittsburgh, PA	Encina	Windows 3.1, NT & 95	AIX; HP-UX; Solaris	X		X
Wayfarer Communications Mountain View, CA	Incisa	Windows NT & 95	Windows NT		X	
XcelleNet Inc. Atlanta, GA	Essentials Session-Xpress	Windows NT & 95	Windows NT		X	

Limited measurements have been made at IDA on an air defense system demonstration based on the SPLICE subscribe-and-publish middleware product, which is marketed by a subsidiary of Thompson C.S.F. SPLICE itself has heretofore been sold only as a component of military command and control systems, primarily naval air defense systems. Some three dozen such systems have been sold to both NATO and third-world countries during the past decade. Various versions of SPLICE have been installed at IDA for almost two years and IDA staff have worked with the developers of the product in defining a demonstration capability that is representative of the type of real-time capability needed for establishing a Single Integrated Air Picture (SIAP). The SPLICE demonstration has allowed an approximate performance comparison to be made with a current CORBA implementation, but more detailed comparisons would be valuable.

From the results of the IDA experiments with CORBA noted in Chapter 2, one expects that, for a pair of Sparc20s with the track management server on one node and the display client on the other, all the resources of both processors would have to be devoted to track movement to move approximately 270 tracks per second between the two nodes. In this case, no resources would be reserved for other functions such as the highly compute intensive task of track correlation on one node and the highly I/O bound task of track display at the other node.

In the case of SPLICE, the situation is different. The SPLICE demonstration includes track management, track correlation, track display, weapon engagement control and a full fault tolerant network management capability. All of the clients and servers of this demonstration can be dynamically moved to any hardware platform participating in the demonstration. In one test, the environment server (that generates signals simulating approaching aircraft and missiles) was placed on one Sparc20 server together with two radar servers (that convert these signals into tracks). The integrated air picture display client and the correlation server were placed on another Sparc20 server. The remaining clients and servers that manage the LAN connections, provide the subscription broker capability, and provide fault-tolerance were also running on these two nodes. The demonstration was then run with 3,600 tracks being generated at all times.

The one radar server simulated a radar that received updates every 1.7 seconds for all tracks. The other radar simulated one that received updates every 1.0 second. Therefore, one server was publishing 1,800 updates every 1.7 seconds over the network and the other one was publishing 1,800 updates every 1.0 seconds. At the other node, these 3,600 updates were being received and correlated, and a local, memory-resident track database was being built by the SPLICE agent at that node. The correlation server was then broadcasting 1,800 correlated tracks per second and the SIAP display client was displaying the 1,800 correlated tracks asynchronously at 1.0 second update intervals from the local track database.

During the running of the demonstration, perfmeters were displayed for each of the Sparc20 machines. The perfmeter on the node that was publishing the track updates (the one with the two radar servers) never exceeded 35% of capacity. The perfmeter on the other node (which included the display client) stayed above 90%. Based on the assumption that receiving the published track updates took about the same amount of resources as publishing them, it was concluded that the overall track movement between publisher and subscriber took about 35% of the processing capacity of the two Sparc20 nodes. Since the perfmeter of the other node stayed above 90%, it was concluded that the major share of resource utilization was consumed by the extensive Xwindows utilization to update the SIAP display and the computation intensive task of correlating the tracks.

Comparing these results with the CORBA results, one sees that the SPLICE implementation could move about 4,650 tracks every 1.0 seconds (raw tracks + correlated tracks) while using about 35% of the processing resource of each node. A presumed CORBA implementation would require all processing resources just to move approximately 270 tracks at the same rate. The conclusion is that the SPLICE implementation is probably some 50 times (4650/0.35/270) more efficient (faster) than a similar CORBA implementation. This comparison is not very exact, however, due to other factors that complicate the situation. First, the track length used in the demonstration was less than half the GCCS track length in bytes. Second, in order to provide a fault tolerant environment and to manage the subscriptions and publications and provide network logging, the demonstration system was publishing many more messages than just track messages. The first factor would argue to lower the factor of 50, while the second would argue to raise it. Nonetheless, the results of the comparison strongly suggest that, for current implementations, the mechanisms of the subscription broker are far more efficient than those of CORBA ORBs in transferring data between distributed nodes.[8]

[8] Measurements for the SPLICE demonstration were also conducted on more capable processors. While analogous measurements do not exist for the CORBA implementation, it is still interesting to note the SPLICE performance figures. In an experiment like the one described above but where the server that hosted the display processor and track correlator was replaced with a single processor UltraSparc, 5,000 tracks could be managed simultaneously. This equates to about 6,470 tracks per second including both raw tracks from the radars and correlated tracks from the correlation server. In another experiment, one radar server was placed on each of two Sparc20 nodes and the correlation server and SIAP display client were placed on an UltraSparc node. In this case, 9,000 tracks could be managed simultaneously, which equates to about 11,650 raw plus correlated tracks per second.

4. FUTURE CORBA PERFORMANCE

4.1 New OMG Initiatives

The previous chapters indicated limitations in CORBA performance, so a natural question is whether improvements are anticipated that would enhance this performance. It turns out that several CORBA services have been proposed that would provide such enhancements. The approach to realizing enhancements is for a Special Interest Group or a Task Force under the Object Management Group (which is the body overseeing the approval of CORBA-related specifications) to develop a Request for Proposal (RFP) for a new service and cause a Technical Committee to issue it. Potential implementers then respond with a proposal for a specification of the interfaces required to implement the given service, and ultimately there is a vote by OMG members on the proposal (or competing proposals). Following this, any vendor who wishes to provide the given service does so according to the approved specification.

In recent years, the user domains (e.g., finance, telecommunications, utilities) represented in OMG have become more explicit in stating their needs. This has led to a number of RFPs issued by Domain Technical Committees and associated responses, some of which have been approved and some which remain under discussion. Several of these RFP responses promise capabilities that are directly applicable to the types of performance enhancements necessary for rapid data transfer in distributed command and control systems. These enhancements are the subject of this chapter. Summary remarks are presented in this section, and each of the relevant new services is then described in a separate section in the remainder of the chapter.[9]

The following table lists the proposed CORBA services. It gives a brief statement of their capabilities and benefits as relevant to distributed command and control systems, plus remarks on the status of the RFP responses and an estimate of when implementations of the services

[9] It is interesting to note that the trend implied in these new services is one of greater constraints on implementations. The first practical versions of the CORBA architecture and implementation repository were specified in terms of their interfaces. Each vendor of a CORBA ORB was given complete freedom of implementation. Shortly after introduction of the specification of this model, it was recognized that interoperability between ORBs required a specification of part of the implementation: an on-the-wire data format specification is required if interoperability between ORB vendors is to be achieved. Thus, the Internet Interoperability ORB Protocol (IIOP) was specified. From that point onward, the OMG has amended the CORBA specification as required, specifying more and more of the implementation details – usually to enhance interoperability or performance.

might be expected. It should be emphasized that these are only "best guesses" for implementation dates. These dates could slip if the time set for vote on a proposed service specification is postponed by the OMG (as has happened in the past when there was still significant open discussion between competing proposals). Furthermore, approval of a specification does not guarantee that vendors will take the steps to implement it, although the market forces that led to the initial RFP and response should be a strong incentive for implementation.

The proposed services listed in the table are grouped in three categories. The first category is services that provide new alternatives for system design. In this category, the notification service and data stream control service seem the most important for enhancing performance of command and control systems because these two services provide, respectively, true subscribe/publish and multicasting capabilities. As seen in the previous chapters, these are key features that lead current subscription brokers to have improved performance compared to currently implemented object request brokers.

The second category in the table is control of resources at a low level. The real-time service noted there should significantly enhance the ability to construct applications with predictable performance deadlines. However, the extent to which this service will also enhance the timeliness of the application execution is not apparent, and that would appear to be the more important requirement from the perspective of command and control applications.

The third category is services that involve a modification to the Object Management Architecture (OMA), which is the basic architecture underlying CORBA. The two services described allow more ready access to the data within objects and to the movement of object data than was assumed in the original OMA. While these two services should be useful, the most significant ones from the standpoint of command and control systems still appear to be the notification and data stream control services noted above.

Given the length of time it takes to develop an RFP, respond to it, and then implement the service, the services listed in the table give a picture of where CORBA is heading for about the next two years. Beyond that, no predictions are made. However, one general trend should be noted. As evident in the services discussed here, OMG appears to be showing significant flexibility in responding to the needs of the user community, e.g., by admitting some modification to the original architecture and allowing greater constraints on implementation if that is what is necessary to improve performance. Thus, it could be that CORBA will continue to undergo modification in the future to enhance its performance if there is significant user demand for such.

Table 3. Proposed CORBA Performance Enhancements[10]

Service	New Capability	Benefits	Number of RFP Responses	Status; Availability of Implementation[11]	Remarks
New Alternatives for System Design					
Notification Service	Enhances current event service by adding channel management (e.g., subscription, filtering, priority, and QoS) and structured events, thereby leading to full Subscribe and Publish semantics.	Prepositions data to allow more timely processing by applications. Filters decrease the use of network bandwidth. Structured events improve message-processing efficiency.	One - in ballot	Undergoing adoption ballot as of 1 May 98. Estimate 2Q 99 availability.	Efficiency is still limited by CORBA's underlying use of request-reply semantics (e.g., precludes multicasting) and required compatibility with event services (e.g., necessitates two-phase push).
Publish/ Subscribe Extensions (preliminary)	Permits transmission to subscriber of additional information about the capabilities of the publisher; extends Publish/Subscribe for use within aggregated components.	TBD.	None - still at proposed RFP stage	The RFP has not been issued yet. No estimate of availability is possible at this time.	Being considered by the Healthcare Domain. Substantial skepticism has been expressed about the need for these extensions.
Control and Management of Audio/Visual Streams	Provides for control of Audio/Visual and (potentially) data streams on a point-to-point and a point-to-multi-point basis, also allowing for various kinds and levels of QoS.	Permits the use of multicast in broadcast of IDL typed data.	One - adopted	Adopted 23 Sep 97. Estimate 4Q 98 availability.	Although not intended specifically for computer data transmission, the specification permits this transmission without requiring CORBA reliable semantics.
Asynchronous Messaging Service	Provides two asynchronous models of method invocation: callback and polling. Allows a client or server to disconnect from an invocation and to later reconnect to complete a response. Provides delivery ordering based on time of origin or priority. Includes a QoS framework and policy hooks.	Increases design flexibility over simple request-reply semantics without introducing substantial complexity for the developer.	One - ready for ballot	Initial submissions have been received. Deadline for revised submissions was 31 Mar 98. Estimate 2Q 99 availability.	

[10]CORBA is used here to denote the ORB as well as other Object Management Architecture components. Cutoff date for status information in the table is 15 May 1998.

[11] Revised submission dates may be altered by the OMG.

Table 3. Proposed CORBA Performance Enhancements (continued)

Service	New Capability	Benefits	Number of RFP Responses	Status; Availability of Implementation[12]	Remarks
Control of Resources at a Low Level					
Mobile Agent Facility	Provides a framework for moving an agent to a remote site and for hosting its operations at that site.	Permits choice of sending a program to the data when that is more efficient.	One - adopted	Adopted 10 Feb 98. Estimate 1Q 99 availability.	
Real-Time Service	Provides greater control of system resources. One proposal achieves this by requiring both applications and ORB to use the real-time operating system interfaces. A second proposal admits the ORB itself to effect this control and exposes the control to the applications developer. (Two of the five proposals were reviewed – they seem representative of the set.)	Could improve the ability of applications to achieve predictable performance of object interactions. (Note: predictability does not necessarily guarantee that desired timeliness goals will be met.)	Five - significant differences	Initial submissions have been received. The deadline for revised submissions is 6 Jul 98. Estimate 2H 99 availability.[13]	Contents of the submissions reflect substantially different views of what is to be provided. All submissions assume that the operating environment includes POSIX RT extensions.
Modifications to OMA					
Multiple Interface and Composition Service	Permits multiple interfaces to an object and direct access to individual object interfaces in a composite object[14]. Provides facilities for assembly of aggregate objects including interface navigation. Provides a mapping to Java beans; improves current mapping to DCOM objects.	Permits direct access to component interfaces in an aggregate for higher efficiency.	Four	Revised submissions have been received. Deadline is 18 May 98. Estimate 2H 99 availability.	May obviate the need for interface definitions to support multiple inheritance.
Objects by Value	Provides return of the value of an object in addition to or instead of an object reference. Provides control of marshalling, buffering, and transmission of data elements transmitted by object value.	Substantially improves the efficiency of transferring data objects from one host to another.	One - passed Technical Committee vote	The submission passed as of 1 May 98. Estimate 2H 99 availability.	

[12] Revised submission dates may be altered by the OMG.

[13] Some of the vendors providing submissions have implemented partial realizations of the technologies.

[14] There are significant differences about how this is to be achieved. Two proposals advocate use of the Component Definition Language and Facility from the BODTF RFP-1. Another proposal advocates the use of a structured design language (OMG-STRUDL). The fourth proposal changes object references to interface references.

Two notes of caution are warranted regarding realizing the performance enhancements offered by the new services. First, existing applications would most likely have to be modified to take advantage of a new service. So there is a time and cost factor involved. In fact, for applications currently in development, it might be best to design them now to use the new capabilities when they become available. Second, a given vendor would most likely not develop its ORB with all the services listed in the table, and in fact might include only one or a few select ones. CORBA interoperability is not such that one is certain services from different vendors can be easily combined. Thus, the command and control system developers would be faced with the choice of choosing the ORB that had the new services most relevant to them.

4.2 Notification Service[15]

The proposed Notification Service [NOT 98] builds on and should provide significant enhancement in capability over the existing Event Service [EVS 97], so it is useful to first describe this latter service briefly. The original CORBA services specification included an Event Service that connects suppliers of events with consumers of events by means of a "channel object." This service is an *M*-to-*N* service with *M* suppliers and *N* consumers. Normal use of the service proceeds in the following manner. An event supplier (server) causes the instantiation of an *event channel*. Other suppliers may use this event channel for the distribution of their events after the initiator sets it up. When a supplier wishes to publish an event, it can "push" the event to the event channel by doing a remote method invocation on the channel object and passing the event data as an input parameter in the invocation. Or the event channel can "pull" the event from the supplier by doing a remote method invocation on the supplier object and retrieving the event as an output parameter in the invocation.

Thereafter, the event channel conveys the event object to the consumer in one of two ways. The consumer either pulls the event data from the event channel via a remote method invocation on the channel object, retrieving the event data as an output parameter in the invocation. Or the event channel pushes the event data to the consumer by doing a remote method invocation on the consumer object. The event channel can be either "typed" or "untyped." In the first case the consumer knows in advance what type of event object will be received. In the second case dynamic tag interpretation is used to determine the type of data being delivered.

[15] Because of the potential importance of the Notification Service to distributed command and control systems, an appendix is included providing more detail on this service. This appendix should serve as a useful summary to the proposed specification, which is over 200 pages in length.

The OMG requirement for compliance with the request-response model of CORBA interferes with the goal of an optimum implementation for the Event Service. This occurs because each participating object in the event transfer process requires that its messages to other objects be acknowledged before the thread used for a remote method invocation can be utilized again in computation. If compliance with the CORBA request-response model were not required, alternatives for reliable service could be constructed using sequence numbers, simultaneous acknowledgment of multiple messages after a period, or other reliability schemes. Because of the requirement for compliance, reliable multicast is not an option in the implementation.

In fact, the existence of the "channel model" itself is a source of communications overhead because it requires two remote method invocations to get a message from a supplier to a consumer. It takes one remote method invocation to get the event from the supplier to the event channel and another one to get it from the channel to the consumer.

The telecommunications community and other industry domains (e.g., the financial and healthcare communities) had the need for more performance and functionality than was available in the Event Service. In addition, these industry domains began employing non-CORBA alternative solutions such as the subscribe-and-publish middleware currently available. The OMG responded by issuing an RFP for a new Notification Service that will subsume and advance beyond the old Event Service. A combined response answers the RFP. In this paper, we concentrate on the Joint Revised Submission of the group of submitters including TIBCO, IBM, Oracle, ICL, and 3 ORB vendors, which was selected for ballot.

The response proposes a Notification Service that provides the full functionality of subscribe-and-publish middleware while at the same time retaining the channel model of the Event Service. This approach provides backward compatibility to the Event Service, while at the same time promising significantly greater efficiency. However, this backward compatibility still could impose some limitations on the efficiency of implementations of the new service. That is, the compatibility requires implementations to be based on the request-response model (e.g., thereby precluding multicasting) and to use two remote method invocations to pass an event. Thus, while significantly faster than the Event Service, the Notification Service is unlikely to transfer data as rapidly as the non-CORBA subscription brokers noted in the previous chapter.

The efficiency of Event Service implementations is limited by the fact that the service definition does not provide for the filtering of events by the supplier. If one event is required, all are sent. Further, all are sent using the DII protocols in which each data item is labeled and any consumer must interpret what it receives. The Notification Service will allow more efficient operation by providing for filtering the event stream to obtain only desired events.

Implementations are being further restricted so a common format for data may be employed, thus substantially speeding the task of conveying data from publisher to subscriber. The response proposes typed and untyped events as in the Event Service, and a new category of *structured events* and sequences of structured events. Structured events provide a more efficient way of filtering values for notification channels with results that are organized in a common format (e.g., stock quotations, fixed format messages, and so on).

The Event Service does not provide for specification of Quality of Service (QoS). The proposed Notification Service permits subscribers to instantiate a filtering object attached to the event channel object which can restrict the flow of events to the subscribers employing that filter. The service also

> "enables each channel, each connection, and each message to be configured to support the desired quality of service with respect to delivery guarantee, event aging characteristics, and event prioritization."

It should be known soon whether the Joint Revised Submission to the Notification RFP will be adopted (it is currently in ballot). In the event of adoption, which appears likely, implementation should follow swiftly.

4.3 Proposed Publish/Subscribe Extension Service

Members of the OMG Healthcare Domain Interest Group have followed the development of the Notification Service initiative and feel it is insufficient for their needs. They have drafted an RFP for discussion by their group. This proposed RFP solicits proposals that permit the transmission of additional information about the capabilities of a publisher to the subscribers. Submissions would need to extend the existing Notification Service response to permit specification of:

publication of events offered/desired;

publication of availability of push/pull services;

publication of the number of connections supported for each style of event supplied and consumed;

the mechanisms for connecting and disconnecting suppliers and consumers of events by type, content, and supplied data;

how producers and consumers can modify the terms of publishing and subscribing;

how the number of connections may be determined as well as endpoint reference;

how control may be established on the number of events delivered in a single cell.

Members of the Notification Service Working Group have expressed substantial skepticism about the need for such extensions. However, the Healthcare Domain Interest Group maintains their claim that the Notification Service specification provides a group of mechanisms without a profile for their use. It remains to be seen whether the OMG will issue an RFP for these publish/subscribe extensions.

4.4 Control and Management of Audio/Visual Streams

The Object Management Architecture (OMA) did not originally address the concept of continuous media as might be used to transmit multimedia streams of information. A response to an RFP [AVS 98], adopted in September 1997, provides an interface that can be used to control and manage such continuous streams of information. The streams are not added as basic elements of the OMA; instead, only interfaces to objects which control or administer these streams are added. Put simply, the streaming data is "outside" the ORB.

The following figure and text from the RFP response by IONA Technologies, Plc, Lucent Technologies, and Siemens-Nixdorf, AG illustrates in more detail how the adopted technology will support the audio/visual stream:

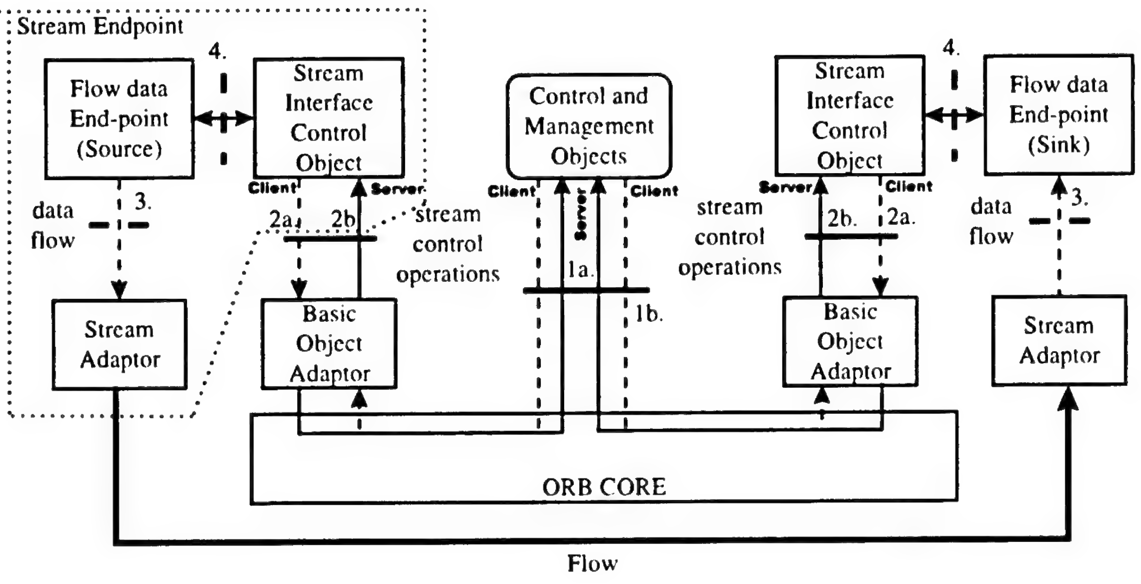

Stream Endpoint
4.
Flow data End-point (Source)
Stream Interface Control Object
Control and Management Objects
Stream Interface Control Object
4.
Flow data End-point (Sink)
Client
Server
Client
Server
Client
Server
Client
data flow
3.
2a.
2b
stream control operations
1a.
1b.
stream control operations
2b.
2a.
data flow
3.
Stream Adaptor
Basic Object Adaptor
Basic Object Adaptor
Stream Adaptor
ORB CORE
Flow

Figure 4. Example Flow from the OMG RFP

"The figure shows a stream with a single flow between two stream end-points, one acting as the source of the data and the other the sink. Each stream end-point, shown as a dotted encapsulation in the figure, consists of three logical entities: a stream interface control object, that provides IDL defined interfaces (as server, 2b) for controlling and managing the stream (as well as potentially, outside the scope of this submission, invoking operations as client, 2a, on other server objects), a flow data source or sink object (at least one per stream endpoint) that is the final destination of the data flow (3), and a stream adaptor that transmits and receives frames over a network.

The Stream interface control object is shown in the figure as using a basic object adaptor, that transmits and receives control messages in a CORBA compliant way. This RFP submission does not require any changes to the BOA/POA or IDL language bindings to accommodate the control of A/V streams.

When a stream is terminated in hardware, the source/sink object and the stream adaptor may not be visible as distinct entities. How the stream interface control object communicates with the source/sink object and perhaps indirectly with the stream adaptor (interface 4 in the figure), and how the source/sink object communicates with the stream adaptor (interface 3 in the figure) are out-side the scope of this RFP submission.

This "Control of A/V Streams" RFP submission provides definitions of the components that make up a stream and for interface definitions onto Stream control and management objects (interface number 1a.), and for interface definitions onto stream interface control objects (interface number 2b.) associated with individual stream endpoints.

In particular, CORBA interface references for stream interface control objects are used to refer to all stream endpoints in parameters to stream control operations defined in this submission. Thus, this submission does not need to define a new IDL data type for stream interface reference."

Continuous media are comprised of streams that may be decomposed into groupings of unidirectional flows, each with its own quality of service parameters. A stream may have flows going in both directions and may have point-to-point or point-to-multipoint flows. Streams usually are employed to deliver information from stream endpoints on one platform to stream endpoints on other platforms.

Quality of service is a property of streams and flows that can be established via interfaces to objects defined by the service. Properties for video and audio are divided into application and network level QoS. Examples of network QoS include ErrorFree, ServiceType, Bandwidth and Bandwidth_Min, Jitter, Delay, and Protection attributes. ErrorFree can be true or false. ServiceType can be best effort, guaranteed, or predicted.

That streams can deliver data is shown in the following quotation from the adopted response:

"So far, streams have always been shown between multimedia devices. It is possible that a programmer will want to stream data between objects in a distributed application, which have nothing to do with multimedia devices. For example, streaming updates of the positions of players in a distributed multiplayer game. Where there is no analogue of a multimedia device involved then asking the application programmer to specify a 'dummy' multimedia device class is inappropriate and an unnecessary overhead. The framework makes it possible to have a stream existing independently of any multimedia devices. The stream will be established directly between two StreamEndPoints which have been created by the application programmer."

The submission proceeds to give the IDL necessary to create such a connection, but does not elaborate further on how the connection is to be used. Using this facility, designers can develop alternatives to the Event, Notification, and Extended Publish and Subscribe services for those instances in which data flows between publishers and subscribers are most appropriately modeled by continuous data flows.

This last fact makes the stream service of particular interest for distributed command and control systems. That is, while this point has not been explored in detail, the service does not appear to be limited in applicability to just audio/visual data, but can potentially be applied to other types of data that can be regarded as approximately continuous in their flow (e.g., the sequence of detection tracks from a radar). Furthermore, the manner of streaming does not have to satisfy the CORBA request-response model, so schemes such as multicasting can be used. Thus, more rapid transfer of data than is achievable with the Event or Notification Services might be possible.

4.5 Asynchronous Messaging Service

While users from various domains represented in the OMG utilized asynchronous messaging on their mainframe computers, this capability was not present in the original OMA or its services. This service is characterized by a one-of-many to one-of-many relationship which is difficult to do in a request-response environment. As a result of the needs of the community, an Asynchronous Messaging RFP was developed, and one response [AMS 98] was generated and is ready for vote. This new service offers additional functionality unavailable with the original paradigm.

Several different semantic models of distributed computing are possible. The OMG selected request-reply semantics between client and server. This model is inherently a one client to one server model. The OMG selected a model that required reliable communication between client and server. Because initial implementations were developed on TCP/IP and because the error detection used in TCP/IP is based on immediate acknowledgement of any message, asynchronous messaging is made more complicated.

The new RFP provides two asynchronous models of method invocation: callback and polling. In the former mode, the client registers interest with the server and provides an object to answer a callback by the server. When the server determines that a reply is to be issued, it calls the specified callback object. In the latter mode, the client polls the server for a response: in this case, the client issues separate requests, the first to get the service, and the second to pick up the result. The same semantics of request and reply with respect to reliability and number of responders are observed as in a conventional remote procedure call (RPC) except that no result is returned to the client from the first call.

In a case where clients may be disconnected from the network, there is a further need. The RFP response provides time independent invocation utilizing store and forward routing of messages. In this case the message from the service is temporarily delivered to a specified "third-party" object until a client (which may be different than the original one) "picks up" the message. This frees the service from waiting for a reply from the disconnected client.

Currently, RPC preserves delivery order. There are situations when time of origin is not the desired delivery order. The RFP response provides delivery ordering based on priority of the request instead. It does this by developing a quality of service framework. The QoS framework provides a set of attributes that may be specified which govern how the asynchronous messaging service is to operate. These attributes may be specified in the policy statement for the domain or may be associated with messages sent by an asynchronous RPC.

Like the Notification Service, the Asynchronous Messaging Service provides new options for using other models of distributed computing than the simple request-response model characterized in CORBA by the remote method invocation. The two models that are likely to be approved in the new Asynchronous Messaging Service (callback and polling) are perhaps close to the original request-response model in that there is still an individual request and an individual response for each client-server interaction. What is different in both cases is that the request and the response actions are separated, the temporal ordering among responses can be changed, and control of when the response is made can either be placed with the client or the server. There is thus greater flexibility in design of distributed systems that can lead to performance enhancements.

4.6 Mobile Agent Facility

With the popularity of Java and the common transmission and use of applets, the need for a standard method of supporting mobile agents was recognized by the OMG. An RFP was issued and its response [MAS 97] approved. The RFP addressed the question of what aspects of the agents should be standardized (at this time) in order to assure interoperability of agents written in the same language and in order to provide a standardized management framework. In evaluating responses, a technology was chosen. This addition forces the standardization of the environment provided to an agent and further reduces choices available to the implementers.

This RFP does not standardize agents. Instead it provides a standardized agent management, tracking, and transport structure. The following are some of the services provided:

agent creation, local and remote;

agent transfer from host to host including serialization and de-serialization of an agent, deactivation on the sending host, and reactivation on the receiving host;

finding agents;

ensuring a secure environment for agent operations;

providing a domain structure for the management of agents operating in a "region";

providing globally unique names for agents, systems of agents, and places of operation.

The performance enhancements possible using the Mobile Agent Facility can be illustrated as follows. In current CORBA implementations the following situation may pertain: large amounts of data may be sent from a server to a client which then operates on this data to produce a result consisting of much less data. Overall performance can be limited in this case by the time it takes to transfer the original data, and the bandwidth so used could degrade the performance of other applications. Sending agents to the source of the data to filter it before being sent can improve performance. Furthermore, if the servers have available processing capacity, the network load can further be reduced by having the agents perform computations on the server on behalf of the client, sending only the computed result to the client. For example, if the client is computing joins on tables supplied by a host, it may be more efficient to send an agent to the host to perform the joins, returning only the final join to the client rather than all the raw data needed to compute the join.[16]

4.7 Real-Time Service

Real-time systems require that tasks be executed in predictable and controllable lengths of time. Typically, the desired executing speeds are "fast", but the emphasis is not on speed per se, but rather on predictability of performance. Designers usually meet this requirement by having detailed, low-level control over the system they are developing. There are two schools of thought as to how such control can be obtained for distributed systems using CORBA. One school believes the POSIX real-time extensions can provide the necessary control. The other school feels it necessary for CORBA to have a separate run-time system with appropriate interfaces to allow control of individual processes, with the inner workings of this separate run-time system being exposed to the application developer.[17] As a consequence, there are two fundamentally different types of response to the Real-Time Service RFP.

The first is a response mandating that CORBA implementations use the POSIX real-time extensions to provide real-time services rather than having a separate run-time system of their own. This type of response envisions the Real-Time Service being a collection of interfaces that allow a user to specify to the real-time broker what type of process scheduling is desired and what the processing environment should be. Then the real-time broker would assure these

[16] Such agents can also fulfill many of the needs expressed in the filtering section of the Notification Service.

[17] In both cases, it is assumed that CORBA is running on top of a POSIX operating system with real-time extensions.

requirements were met on behalf of the user by means of utilizing the POSIX real-time extension interfaces.

The second type of response assumes that CORBA implementations will have a separate run-time system of their own. Such a system would monitor and control executing processes; it can be regarded as standing between the processes and the operating system. This type of response envisions the Real-Time Service to be a collection of interfaces that allow the user to take command of this run-time system for the purpose of explicitly controlling the process scheduling and the processing environment.

Of the two types of response, the one assuming that CORBA implementations will utilize the POSIX real-time extensions is by far the smaller because the collection of required interfaces is much reduced. This type of response assumes that all that is needed is a way of communicating process scheduling needs and real-time processing environment needs to the real-time broker so it can enforce these policies and environmental conditions. In many ways this is the more elegant specification.

The other type of response envisions essentially duplicating many of the POSIX real-time interfaces in the CORBA Real-Time Service specification. This is because of the need to manipulate the CORBA run-time system via these interfaces just as the POSIX run-time system would be manipulated using the POSIX real-time interfaces. As a result there is an extensive collection of real-time interfaces to control just about anything the operating system might normally do in controlling processing and storage resources. This is because it is assumed that with a real-time broker, many of these normal operating system functions will now be performed by the CORBA run-time system.

There are currently five different responses [AAR 98, HCVS 98, IN 98, LMFS 98, OIS 98] to the real-time RFP. Probably the most significant response utilizing the existing POSIX real-time interfaces instead of having a separate CORBA run-time system is the response from Iona Technologies and Northern Telecom [IN 98]. Their response is based on their work with developing a real-time broker capability that Northern Telecom is currently using in some of its products. Probably the response with the most industry support weighs in on the other side of the argument. The joint responders are: Alcatel, Hewlett-Packard, Lucent Technologies, Object-Oriented Concepts, Sun Microsystems, and Tri-Pacific [AAR 98]. This response was developed in collaboration with the following additional organizations: Deutsche Telekom, France Telecom, Humboldt University, Mitre, Motorola, and Washington University. This response specifically states the following as the current problems with CORBA for real-time applications:

> "The architecture does not provide interfaces to specify end-to-end QoS requirements which are necessary for real-time applications. Furthermore, the opaqueness of the ORB core concerning resources and the full invocation path makes end-to-end QoS enforcement hard to meet."

> "Lack of real-time programming features such as asynchronous invocations, timed operations and transport layer flow control notification."

> "Lack of performance optimizations lead to significant throughput and latency overhead. Improvements in data copying, message buffering and de-multiplexing are required to allow many performance sensitive real-time applications to use a CORBA ORB."

Having said all the above, however, it is not apparent that the Real-Time Service will be as critical a development for distributed command and control systems as, say, the Notification Service. This is because the Real-Time Service focuses on guaranteeing predictable performance, but not necessarily on increasing performance speed, although it could have benefits in that regard. Thus, the Notification Service will probably do much more for the rapid delivery of data among the distributed components of a command and control system. Still, the Real-Time Service could be useful in cases where it might be necessary to control the execution of components in a highly coordinated manner (e.g., battle management assets on a local area network).

4.8 Multiple Interface and Composition Service

In cases where software components are aggregations of lower level components, performance is hindered if there is only a single interface to the component. If an interface in a lower level component is to be accessed, the call would have to go to the interface of the aggregate component and thence be directed to the lower level component. The situation is even more complicated if the aggregate component is composed of components that are aggregates themselves, and so on. Clearly, being able to make calls directly to the interface of the lower level component would enhance performance. Currently, the OMA allows only specification of a single interface to an aggregate component.

Not only is a single interface sub-optimal from the performance point of view, it is also sub-optimal from the development point of view. It is true that the single interface could inherit from more than one class of objects in order to form the interface. For example, an object could inherit from a computational interface, a management policy interface, and a security interface. But this combination of interfaces by multiple inheritance would be ungainly for developers – if one of the pieces changes, the whole interface must be re-compiled. Further, new interfaces to the object could not be added without affecting the old ones.

To redress this situation, an RFP was developed focused on the needs for multiple interfaces and composition services. In one submission [DTSC 97], an extension to the specification language called OMG Structural Definition Language (OMG-STRUDL) is defined which is a superset of current IDL. This submission claims to have several advantages as summarized in the following:

> "The specification language syntax and related technology to support object assembly are relatively mature. Analogous syntax and technology has been developed and used within the Telecommunications Information Networking Architecture Consortium (TINA-C), and within TINA-C member companies, over a number of years.
>
> The multiple-interface (assembly) concept specified in this submission has been used extensively in TINA-C specifications of scalable telecommunications systems and components.
>
> OMG-STRUDL assists with the management of complex version relationships, with the control of client sessions and with the specification of close relationships between multiple services provided via interfaces specified in IDL.
>
> The full TINA-C Object Definition Language (TINA-ODL) includes a number of features which are of interest to the OMG, including stream interface support and mechanisms for the specification of Quality of Service on both stream and operational (IDL) interfaces. Although these features of the language do not form part of this submission, this document does contain specifications, which allow the inclusion of support for important additional capabilities through straightforward extensions to the grammar. This has important implications for the retention of backward compatibility in the OMG specification language and for associated tool sets.
>
> No changes to CORBA 2.0 compliant applications are required by this submission."

A second submission [IBM 97] seeks a minimal change oriented toward the use of Java Beans. It requires extensions of CORBA to account for the differences in Java and CORBA semantics.

Two additional submissions [DA 97, GI 97] attack the problem from a different perspective. Both are based on work being done for the Business Object Domain Task Force (BODTF) and are based on work described in the BODTF RFP-1, which is undergoing a fax vote in the Domain Technical Committee. These proposals take the view that the aggregate component is but an abstraction of its components. The structure of the aggregate is described in component definition language (CDL) and maintained in the component definition facility (CDF) as metaobject information. These proposals apparently do not value the delivery of pre-packaged aggregate components to users since they maintain all aggregate level information only in the metadata, which developers of packaged components would not wish to expose to purchasers.

All proposals would allow direct access to the base object interface, but would use very different techniques for doing so. It remains to be seen which proposal will be accepted. This area is particularly important and vendors have a strong vested interest in having their RFP be successful. The deadline for final submission has already been extended once and may be extended again if compromise cannot be reached.

4.9 Objects by Value

The OMA currently only permits the transmission of an object by object reference. In this paradigm, an interface can return an object reference to any instance of an object that the interface controls. To effect an operation on that object, the client receiving the reference makes a request of the service utilizing the object reference and invoking the desired method. No notion of providing the value of the object across the network exists (except for the externalization service). Often such use is very inefficient when a copy of the object is desired. In order to get the state, multiple calls to the service might be required to obtain the entire state, which would then be packed into an object controlled by the client.

CORBA permits the passing of values with a request for service and the return of a single value. These values can be basic values or they can be compositions of values or structured items. One of the basic values is an object reference, but, as noted, values of entire objects could not be transmitted. The reason for this decision is that CORBA specifications could be made independent of the programming language used for implementation. However, using standard CORBA operations is very inefficient when used to copy an object from machine to machine when the objected consisted of data elements and no methods.

A significant difficulty is found in the use of CORBA when many object values need to be moved from one site to another. As a result, a compromise implementation is often adopted. All objects are converted to strings of characters. The characters are passed using sequences of characters. This proves much more efficient despite the additional work required converting the objects to strings at the server and from strings to an object value at the client.

The need for copying objects from server program to client program was high enough that an RFP was issued to perform this operation in one call. Because this cannot be done in a portable manner without specifying the on-the-wire implementation of the object, a new protocol was developed so that this would be possible. The new protocol further constricts the class of implementations but can greatly improve the efficiency of performing the operation.

This RFP response [OBV] has been approved by the committee and is undergoing vote. It will likely be approved by the voting list and submitted to the board for ratification. The goals of this proposal are to:

"provide a simple and very robust model that builds on existing CORBA semantics;

respect the current CORBA model that distinguishes, in its type system, interface types from constructed data types;

minimize changes to IDL;

maintain flexibility in ORB implementations while guaranteeing interoperability and portability;
expose complex implementation states in a language independent manner;

guarantee consistent semantics and use across languages;

support passing of objects by value between clients and servers implemented in different languages;

provide natural and convenient support in Java and C++."

The RFP added a new basic value – the value of an object, which is the state of the object. In addition it added the *abstract interface* as a parameter and as a result. The abstract interface is a placeholder for either an object reference or an object value.

The effect of the changes is to constrain implementations since the service and the client must be prepared to receive either an object value or an object reference. This must be determined dynamically for every request/reply. In order to efficiently marshal the actual value to be transmitted, CORBA has been modified to expose the marshalling interface to those who desire to do custom marshalling. In addition, because the sequence of data in an object is likely to be very long, additional options have been added allowing the data to be divided into chunks for transmission. The specification now permits user control of the division into chunks and the transmission of each. CORBA has also been modified to permit factories for object values to use the transmitted object value to create a new, unshared object at the server/client. The RFP specifies changes to the C++ and Java Mappings to permit use of the new functionality, but does not provide language binding for languages that do not support objects such as COBOL, C, and FORTRAN.

5. RECOMMENDATIONS FOR DARPA ACTION

Implications for DARPA action follow from the discussions of the previous chapters. In particular, three general recommendations are made:

- That DARPA ISO form a performance testing program to address data transfer effectiveness in its C2 applications more explicitly. A performance working group would coordinate the activities of this testing program across the relevant ISO programs.
- That the DARPA ISO community become more active in tracking and influencing the actions of the OMG relating to CORBA data transfer performance.
- That the DARPA ISO C2 programs begin addressing how their applications will need to be modified to use the new CORBA capabilities.

Section 5.1 below presents the general rational for the performance testing program and the performance working group, while Section 5.2 describes the activities in the testing program. Section 5.3 elaborates on the involvement in OMG activities and Section 5.4 discusses accommodating the new CORBA capabilities in C2 applications.

The recommendations above focus on data transfer, in keeping with the scope of this paper. However, in considering implementation of the recommendations, DARPA might want to address performance more broadly (e.g., overall C2 application performance) and establish involvement in OMG activities extending beyond performance matters.

The recommendations are directed at the DARPA Information Systems Office (ISO) since that is where most of the DARPA programs in C2 information systems fall. This is meant to include the DARPA-DISA Joint Program Office (DDJPO) since they can have a significant role to play. Furthermore, the Information Technology Office might be involved to some extent, particularly if such issues as changing the fundamental mechanisms underlying CORBA are addressed.

5.1 Establishment of a Performance Testing Program

Timely and efficient data transfer is critical to the functioning of command and control (C2) systems. If the data transfer rate is too low, some functions of the C2 systems might not be realized or readily usable. Thus, achievement of effective data transfer means is of major importance to DARPA C2-related programs. Many of these programs (in ISO) intend to use a

CORBA ORB to effect data transfer, as is consistent with the specification of the AITS architecture to which these programs should conform. But as noted in Chapters 2 and 3, very little quantitative performance data on brokers exists. Furthermore, limited discussion with the ISO programs did not reveal a major emphasis as yet on determining and ensuring adequate data transfer performance.

More explicit attention to data transfer performance therefore appears warranted. It is thus recommended that DARPA ISO form a performance testing program to address this matter. In general, the program would involve determining the data transfer requirements to be met for the particular C2 applications and measuring the performance of the data transfer means to be used. Such a testing program would involve the ISO programs both individually and collectively. Each program would be responsible for determining the data transfer requirements for its particular application. Some of the programs might take a greater responsibility in carrying out the actual testing measurements. In addition, some of the testing activities could be carried out in the DDJPO. Furthermore, a coordinating body – the performance working group – should be established to give impetus to the testing program and to share its results across the ISO programs. This working group would be formed from the development teams for the relevant ISO projects.

5.2 Nature of Performance Testing Program

Generally speaking, there are two different aspects to the performance testing – product testing and system testing. The former refers to testing the given product (e.g., a particular ORB) largely by itself, with only those components (e.g., a client and server) necessary to complete the measurement arrangement. The latter refers to testing the product in a system context in some way approximating the intended application. Both forms are necessary. The product testing provides fairly detailed information and aids in determining the product to be chosen for the application. The system testing is less detailed in terms of examining product behavior, but allows the product performance to be assessed in an actual application context. In actuality, there might be a spectrum of tests from one simply involving the broker and simple drivers up to one approximating a full-up system.

Product Testing. Within the context of this paper, two types of products would be tested – CORBA-compliant ORBs and subscription brokers. In a more general context, other middleware products might be tested, too. The IDA experiments showed significant differences in CORBA ORB performance (e.g., a factor of three in the rate at which remote methods could be invoked). Thus, product testing should involve a significant sample of ORBs and subscription brokers from different vendors. Furthermore, these products are evolving at a rapid pace, so it will be necessary to keep current with updates of these products. In particular, the new capabilities discussed in Chapter 4 (e.g., Notification Service) should be tested as they become available.

Significant attention should be given to determining and specifying the experimental conditions under which a product will be tested (e.g., platform and operating system version, network characteristics, size of data packets transferred, number of packets transferred per ORB request or subscription broker push). This is important so that meaningful comparisons can be made between the results of different tests, especially when dissimilar products (e.g., a CORBA ORB and a subscription broker) are involved, where the underlying implementation mechanisms can be quite different.

In a similar manner, it will be useful to "instrument" the tests to find out the processing times of the various components involved (e.g., marshalling, network transfer) in the overall data transfer operations. This will allow the pacing processes to be identified. This deeper understanding of the performance mechanisms of the brokers will aid in understanding the most efficient and effective ways in which they can be applied. For example, knowing the pacing processes, one would design the overall system to minimize their effect.

Furthermore, these instrumented tests comparing the products of different vendors should give fundamental insights into the structure and underlying mechanisms used in implementing the ORBs and related services. While this might relate more to research than testing, one could seek to extend that knowledge to predict what are the upper limits of CORBA performance, given the current underlying request-reply model is maintained, and what is the likelihood of achieving such limits.

System Testing. System testing involves using a broker in a system configuration approximating the intended application. For example, the broker could be mediating data transfer among a variety of clients and servers to fulfill some C2 functions. To simplify the system tests, some of the components forming the system (but typically not the brokers) could be simulated. In fact, there could be a range of tests where the simplest ones mostly used simulations and the more extensive ones moved to prototype implementations of the components. Care must be taken, however, to understand whatever uncertainties are introduced by the simulations and prototypes. A performance model of the individual components (e.g., data server) is necessary in order to build a simulation of it. Developing this performance model could be a significant undertaking in itself.

Monitoring system performance is of major interest in many application areas, non-defense (e.g., telecommunications) as well as defense. Some tools have been developed for such monitoring (e.g., of network performance). Thus, as an initial step in the DARPA performance testing program, a survey of commercially available tools should be conducted and the appropriate tools acquired and installed. Some of the more research-oriented tools (e.g., Rapide, Wright) should also be acquired. These research tools are more directed at the "logical correctness" of an implementation (e.g., avoiding situations that could lead to deadlock) than to performance per se, but they do have a useful role in checking the overall behavior of a system.

The ultimate end of the system testing is to compare the observed performance to the required performance. If the requirements are not met, a variety of changes to the system might be necessary – use of a different broker, redesign of system components (e.g., a server), or redesign of the overall system (e.g., the partitioning of tasks among components). Obviously the earlier the system performance assessment can be made, the better, so as to avoid committing to expensive development efforts that would later have to be undone or discarded because of inadequate performance.

The system testing, in conjunction with the detailed testing of individual products, should lead to insight about the best way to use the brokers in overall system design (e.g., how data should be aggregated for transmission, how processes should be allocated across computing platforms, as well as more general matters like the functions for which ORBs or subscription brokers would be most appropriate). Thus, in addition to explicit performance determinations, another benefit of the testing program is determination of guidelines for system design. This information should be captured and circulated among the ISO programs. The guidelines could differ significantly across the classes of C2 applications considered.

5.3 Involvement in OMG Activities

The discussion of Chapter 4 made clear that the CORBA community is responding to demands of its customer base by working to enhance the performance-related functionality of its ORBs in several aspects, including ones that relate to the needs of the DARPA ISO C2 programs. Currently, however, the DARPA community has only very limited interaction with the CORBA community. Thus, its is recommended the DARPA community become significantly more involved with the CORBA community. This interaction would be both to understand the new CORBA capabilities being proposed and to influence the direction of such capabilities to meet DARPA needs. Furthermore, the stressing environment presented by the DARPA programs might be regarded as a "laboratory" in which to test the new CORBA products while they are being developed. In addition, implementations of certain services developed within the DARPA ISO programs might be provided to the wider CORBA community for their use.

The CORBA community operates through a set of working groups under the overall auspices of the Object Management Group (OMG). The number of working groups is quite large, even just that subset that has immediate performance implications. The best approach, given the relatively limited personnel resources of the DARPA community, would in general be to track the activities of the groups through their postings (there is extensive exchange of information through electronic mail and web sites) and then commit to personal attendance at working group meetings only in the highest priority cases. The performance working group can be used as a way to coordinate the DARPA involvement in CORBA activities. Some degree of formal agreement between DARPA ISO and OMG might also be necessary.

5.4 Accommodation of New CORBA Capabilities

As described in Chapter 4, a variety of new CORBA capabilities are expected that can significantly enhance C2 application performance. These capabilities – for example, subscribe and publish data transfer and the ability to do asynchronous messaging – provide new design concepts for CORBA applications. Thus, taking advantages of these new capabilities means redesigning the client and server components of the application, as well as installing the ORBs with the new capabilities.

Current DARPA CORBA-based C2 applications are most likely designed to the current versions of CORBA lacking these new capabilities. Thus, to take advantage of these capabilities, it is recommended that redesign efforts, some possibly large in scope, be considered and undertaken as warranted by the requirements of the C2 applications. Given that the new CORBA capabilities might be expected in the relatively near-term (roughly 6-18 months), the redesign efforts might begin soon. As feasible, the required new interfaces could be built into applications being developed now so that the new CORBA capabilities could be used immediately when they become available. However, risk assessments should be made – that is, early commitment to redesign should be based on an assessment that the new capability will materialize and provide significant benefit. The performance working group can serve as a vehicle for sharing knowledge across the programs about the benefits and methods of these redesign activities.

APPENDIX A: NOTIFICATION SERVICE

The Request for Proposal (RFP) for the Notification Service requires it to be backward compatible with the currently existing Event Service. In fact, the proposal calls for inheriting the Event Service objects in the construction of the new Notification Service. Thus, the Event Service is first discussed below before turning to the Notification Service.

A.1 The Event Service Model

The fundamental idea behind the CORBA Event Service model [EMS 97] is to use remote method invocations (RMIs) to push data rather than request (pull) data. Rather than having a client process make an RMI to a remote server to request a data object, the server makes an RMI to the client process as is shown in the following figure. The client's method that is invoked receives the data and either returns a confirmation or returns nothing.

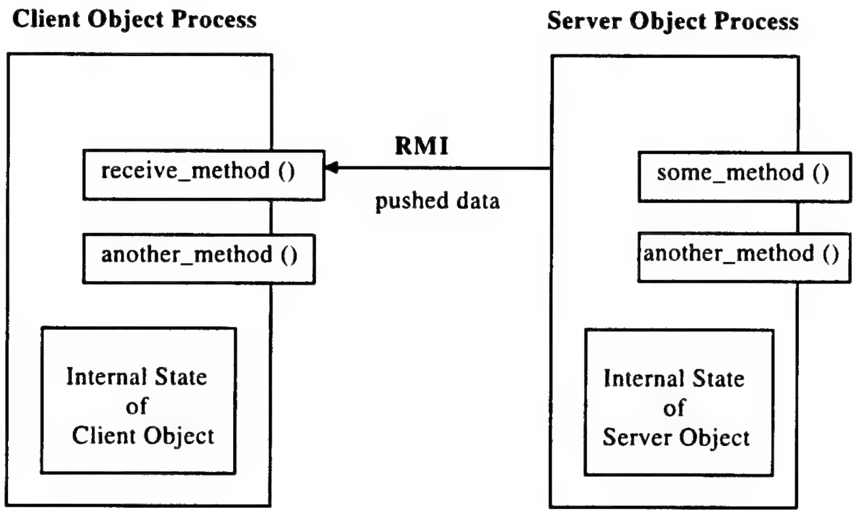


Figure A1. Single Server Invoking Client Method to Push Data

Typical situations involve several different servers making updates to a given class of data objects and many client processes distributed over a network that need access to this class of data objects. To handle this situation with CORBA, but without the CORBA Event Service, would require that all the clients and servers have direct connection with each other. This is an order n^2 connection problem where n is the aggregate number of clients and servers. This causes all the servers to compete with each other for all the client processes' resources to push data updates to them. On average such a scheme can cause large random delays in moving data objects from servers to clients as the servers queue up in the clients' input queues.

To get around this problem, the CORBA Event Service introduces the concept of an *event channel*. An event channel is a CORBA object that is associated with a specific class of client and server objects and sits between the clients and the servers as shown in the following figure. All the clients and servers connect to the event channel rather to each other, thereby reducing the connection problem to an order n problem rather than an order n^2 problem at the cost of requiring two RMIs to move a data update from a server to a client.

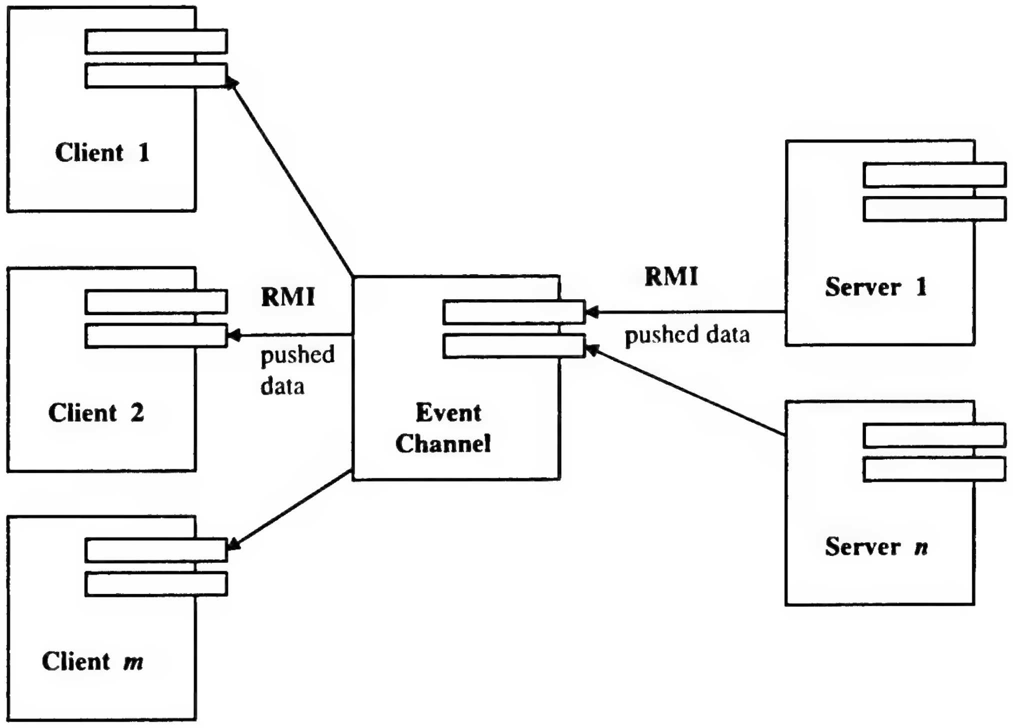


Figure A2. Event Channel Connecting m Clients to n Servers

This, of course, could make the event channel a bottleneck for data transfer, but the effect is not as bad as it might seem at first glance. Although it slows down the data transfer speed by at least a factor of two, it reduces the number of individual competitions for client resources dramatically and the various implementations of this CORBA service can opt for multi-threading optimizations in the event channel. Furthermore, it introduces a more orderly distribution of data updates from servers to clients by translating the n^2 problem into two simpler order n problems, namely of moving the updates from the servers to the event channel and then delivering the updates from the event channel to the clients.

Thus, the Event Service can lead to significant enhancements for CORBA in rapid data transfer. However, this data push performance is, in general, likely to be significantly below that of currently available subscribe-and-publish middleware products for two reasons – namely, the filtering and multicasting capabilities available in these latter products.

With the CORBA model of data push, including its realization in the Event Service, the server decides what to push to the client. Unless there is a predetermined agreement between the client and server of what and how much data will be sent to the client, it is possible for the server to push the client more data than it can respond to, or more data than it needs. The net effect can be reduced performance. A dynamically adjustable filtering scheme that would send just the data the client needs would be preferable (and would in fact simplify design as well as improve performance). The Notification Service will provide this capability.

Multicasting can also enhance data push performance by sending the data to multiple clients in one operation. But this is not allowed by the CORBA standard, which requires acknowledgment of each message sent at the transport layer in order to guarantee reliable RMIs. Most multicast and broadcast protocols are not guaranteed to be reliable. The Notification Service will also be subject to this limitation since it is also required to have reliable RMIs.

A.2 Notification Service for CORBA

The discussion of the Notification Service is based on the Joint Revised Submission [NOT 98] currently up for adoption vote. This proposal was submitted by the following companies:

1. BEA Systems, Inc.
2. Cooperative Research Center for Distributed Systems Technology (DSTC)
3. Expersoft Corporation
4. FUJITSU LIMITED
5. GMD Fokus
6. International Business Machines Corporation (IBM)
7. International Computers Limited (ICL)
8. IONA Technologies, Plc.

9. NEC Corporation
10. Nortel Technology
11. Oracle Corporation
12. TIBCO Software, Inc.
13. Visigenics Software, Inc.

One of the members of this consortium, TIBCO, is currently a major vendor of subscribe-and-publish middleware. Several of the other members of this consortium have endorsed or use the TIBCO product, which is called TIBNet (TIB stands for "The Information Bus"). The consortium proposal is essentially based on the TIBNet product, modified to be compatible with other CORBA Services.

The consortium proposal offers two architectural diagrams in the first few pages of their proposal to help explain how their proposal for the Notification Service is an extension of the existing Event Service and how current Event Service clients could use the new Notification Service without change while getting the benefit of some of the new features. They could, of course, not take advantage of all the new features without making modifications to the client. These two diagrams are shown below.

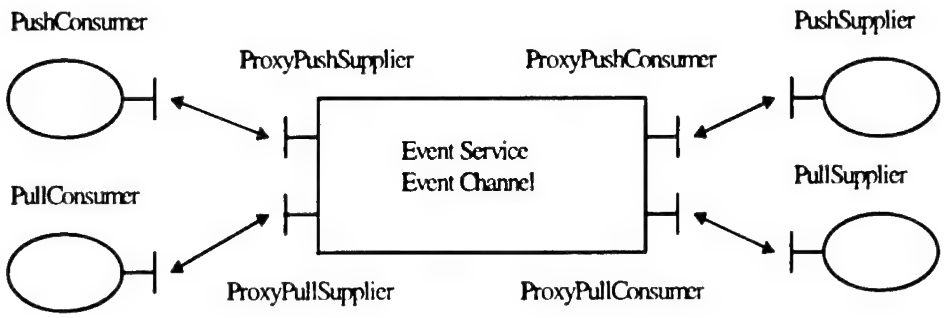


Figure A3. CORBA Event Channel

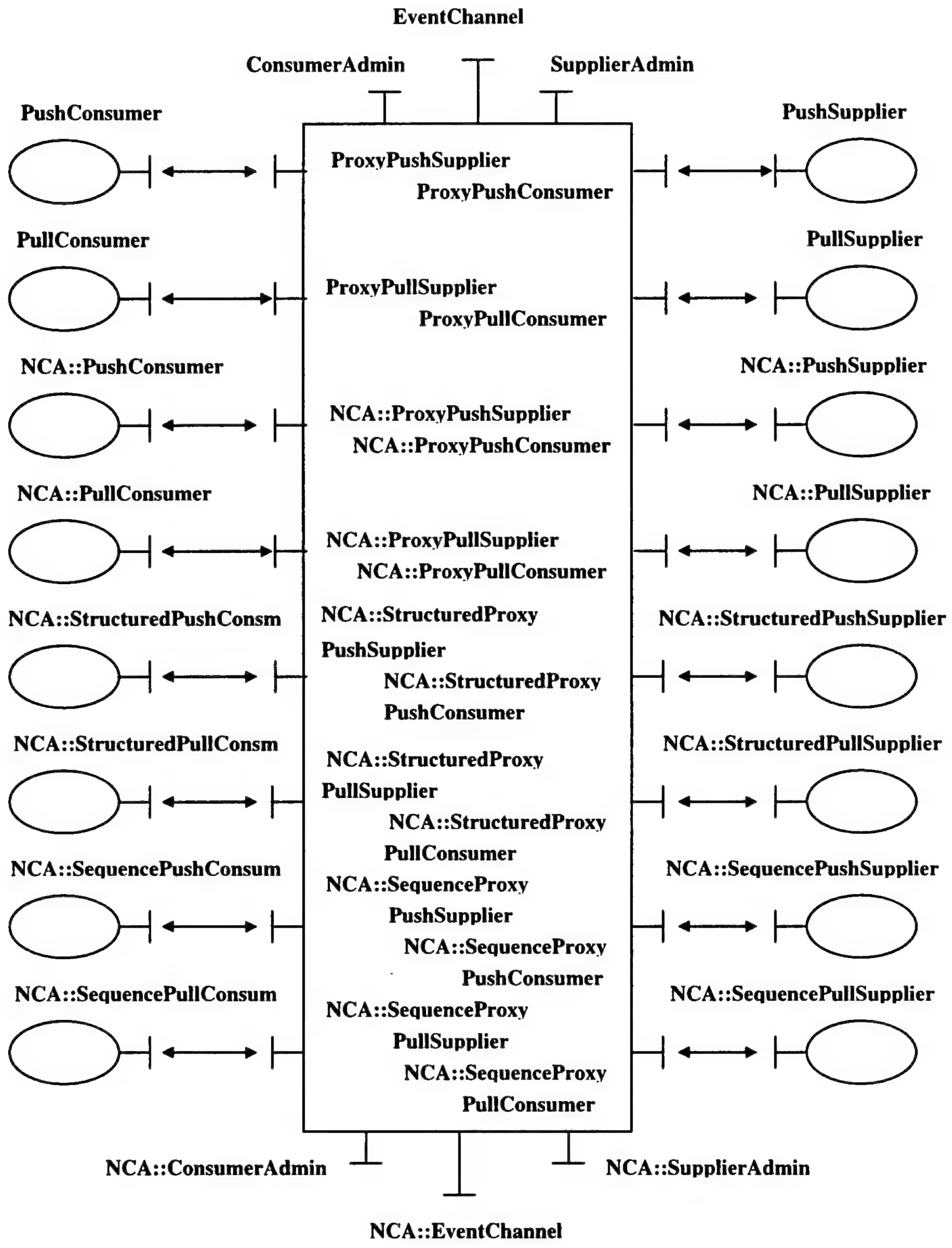


Figure A4. Notification Service Event Channel

Figure A3 is the OMG diagram for the Event Channel that we have already discussed earlier in this chapter. Figure A4 is a very busy diagram whose top section resembles that of Figure A3. It is all the additional parts of the diagram that we now need to explain. The explanation that will follow is a significantly simpler explanation than that given in the 215 page proposed specification. This explanation will concentrate on the significant architectural differences at the expense of a detailed explanation of the various data filtering capabilities.

The Notification Event Channel shown in Figure A4 inherits from the Event Channel object of the Event Service and therefore it can be used as an Event Channel by existing Event Channel clients without reprogramming. But the Notification Event Channel adds much more to the Event Channel in the area of functionality that supports a "smart push" style of distributed architectural design. In fact, the Notification Service provides full subscribe-and-publish syntax and semantics for the Notification Event Channel.

This allows the Notification Event Channel clients to subscribe for precisely the data objects they require and no more. On page 35 of the proposed specification it states "Undoubtedly the most important enhancement of the OMG Event Service introduced by the Notification Service is the enabling of each client to subscribe to the precise set of events it is interested in receiving. This feature is supported in the form of *filter* objects, each of which encapsulates a set of one or more constraints specified in a particular constraint grammar."

This ability to filter the data objects by the clients of the Notification Event Channel occurs at multiple levels. Like the CORBA Event Service *Consumer Admin* object and the *Supplier Admin* object, the Notification Event Service provides for Consumer and Supplier Admin objects but the Notification Channel provides for multiple Admin objects per channel. These Admin objects are essentially object factories that create the Proxy objects, shown in Figures A3 and A4, to which client and server objects will connect to the channel. Each Admin object "manages" the Proxy objects it creates (as will be explained in more detail below). Filter objects can be applied at the Admin object level which effectively creates different classes of clients (the ones that inherit from a particular Admin object) that inherit their basic data filtering profile from that Admin object. The filters can also be applied at the channel level so that all clients inherit default filtering of data. Finally, the filters can be applied at the individual proxy supplier and consumer objects so that a given client can have a unique data filtering profile. Filters at a lower level override filters at a higher level thereby giving the clients the capability to override server decisions of what to send them.

The Notification Services also introduces the concept of Quality of Service (QoS) properties that can be set at the channel level, the Admin object level, the Proxy level and finally at the individual message level.

The notification Service provides a *factory* interface, EventChannelFactory, for creating new instances of notification channels. At the time a channel is created, the client can specify the administrative and QoS properties it requires. There are certain standard administrative and QoS properties that each instance of a channel must support as a minimum. The standard administrative properties include the maximum number of consumers and suppliers that can be connected to the channel simultaneously.

A.2.1 Proxy Objects

In the earlier discussion of the Event Channel provided by the CORBA Event Service, we did not discuss the Proxy objects that are an integral part of both the Event Channel and Notification Channel objects. In Figure A3 we see that an Event Channel has Proxy objects. With an Event Channel the Proxy objects are essentially transparent to the client (consumer) and server (supplier) objects that use the channel. With a Notification Channel, the existence of the Proxy objects is no longer transparent because the Proxy objects have filter objects associated with them that the clients connected to the channel can use to specify their subscriptions to various classes of data objects.

In the case of an Event Channel, the Proxy objects exist for the purpose of allowing the channel to "expand" as more consumers and suppliers connect to the channel. For each new connection, appropriate Proxy objects are instantiated for the supplier and consumer objects to call or for the channel object itself to call. While this is not the only way an Event Channel could have been specified to achieve the same functionality, this design style is consistent with the CORBA architecture. This design style has been preserved in the specification of the Notification Channel. Moreover, the designers of the Notification Channel have decided that these Proxy objects are a good place to locate the filters that are used for precisely specifying subscriptions.

The existence of all these Proxy objects give the impression of formidable complexity to the Notification Channel objects at first glance. But while it is a little cumbersome to explain, we will see that these Proxy objects and their functions are really not as complicated as might first appear and that they might eventually even seem "natural" within the CORBA architectural setting. There are essentially only four different types of Proxy objects. All the different Proxy objects shown in Figure A4 are simply variations on one of these basic types, and the reader will soon learn to recognize which variation is associated with which basic type.

There are two basic types of Proxy objects for data-push and two basic types for data-pull. The basic Proxy objects for data-push are the *ProxyPushConsumer* and the *ProxyPushSupplier*. These two basic types exist in Event Channels. The ProxyPushConsumer object in an Event Channel is there to provide an interface for a supplier (server) object to call, i.e., to connect to. Each different supplier has a different ProxyPushConsumer interface to connect to within the channel. The ProxyPushSupplier object in an Event Channel is there to provide an interface for a consumer (client) object to call. Each different consumer has a different ProxyPushSupplier interface to connect to within the channel.

Event Channel Proxy objects can handle either event (data) objects of type *Any* or typed event objects. For simplicity we will restrict this discussion of Proxy objects to untyped event objects. The Notification Service, while retaining the typed event object capability, offers the concept of *structured* events that is more natural for subscriptions. We assume that users of the Notification Service will opt for structured objects as opposed to typed objects.

Notification Service style Proxy objects differ somewhat from Event Service style Proxy objects. First, Notification Service style Proxy objects are further subdivided into three categories that provide the capability to handle (1) event objects of type Any (untyped events) (2) structured events and (3) sequences of structured events. This gives rise to the four different variations of each of the two basic data-push Proxy objects. The variations of the ProxyPushConsumer objects are:

1. ProxyPushConsumer object
2. NCA::ProxyPushConsumer object
3. NCA::StructuredProxyPushConsumer object
4. NCA::SequenceProxyPushConsumer object

The three variations (2 through 4) are prefixed by NCA: which indicates they are within the scope of the Notification Channel Admin (NCA) object. Similarly the ProxyPushSupplier object has the four different variations which are:

1. ProxyPushSupplier object
2. NCA::ProxyPushSupplier object
3. NCA::StructuredProxyPushSupplier object
4. NCA::SequenceProxyPushSupplier object

In both cases variation 1 is simply the ProxyPushConsumer or ProxyPushSupplier object that is inherited from the Event Channel object. We now explain these three variations of these basic Event Channel Proxy objects. The first ones (the variations No. 2) provide the same capability as the untyped Event Channel Proxy objects but in addition they have filter objects associated with them that can be used to specify what data (event) objects get forwarded (or are forwarded by) the consumer or supplier that is connected to the channel by means of these Proxy objects. Furthermore, the consumers can specify QoS parameters to these Proxy objects that govern how data objects are to be treated with respect to priority or reliability.

The following quotation from the proposed Notification Service specification on page 26 gives some of the rationale for these variations. "Note that by dividing the Notification Service style Proxy interfaces along the lines of the form of message they are capable of transmitting, and by defining separate client interfaces (in the CosNotifyComm module) for clients that deal with each specific form of message, clients of the Notification Service have the freedom to implement consumers and suppliers that deal with events in only the specific format(s) they are interested in sending and receiving them. For instance, to develop a consumer application which only receives events by push-style communication in the form of Structured Events, the developer simply needs to implement the CosNotifyComm::StructuredPushConsumer interface, which only supports a push operation which receives events in the form of Structured Events."

The next variation of basic Proxy objects (variations No. 3) are variations that provide the capability to send and receive Structured Events. The rationale for proposing these Structured Events is given on page 29 of the proposed specification in the form of a commentary of the OMG Event Service. "The OMG Event Service supports two styles of event communication: untyped and typed. Untyped communication involves transmitting all events in the form of Anys. While untyped event communication is generic and easy-to-use, many applications require more strongly typed event messages. To satisfy this latter requirement, the OMG Event Service defines interfaces and conventions for supporting typed event communication. Unfortunately, many users have found typed event communication as defined by the OMG Event Service difficult to understand, and implementors have found it particularly difficult to deal with."

"For these reasons the Notification Service introduces a new style of event message: the *Structured Event*." According to the proposed specification, Structured Events provide a well-defined data structure into which a wide variety of event types can be mapped. Essentially a Structured Event is a standard data structure in which a wide variety of event messages can be stored.

A.2.2 Structured Events

Figure A5 below shows the format of a Structured Event. It is comprised of two main components, namely a header and a body. The header is further decomposed into a fixed part and a variable part. The rationale for this two part header according to the proposed specification was to minimize the size of the header that is required in every Structured Event message in order to enable lightweight messages where the overhead of supplying an additional header field is considered less desirable than any benefit achieved by supplying them. The fixed part of the event header is comprised of three string fields present in every Structured Event, the *domain_type* which identifies the vertical industry domain in which the event type is defined, the *event_type* which categorizes the type of event uniquely within the domain, and the *event_name* that uniquely identifies the specific instance of event being transmitted.

The variable part of the header is comprised of a list of zero or more *name-value pairs*. The "ohf_" prefacing each of these name-value pairs stands for "optional header field." Each of the ohf_name fields is a string and each of the ohf_value fields is of type Any. Inclusion of these fields is optional. The specification standardizes a set of well-defined header field names and defines the data types of their values. These *standard optional header fields* contain per-message QoS information. There are six standard optional header fields that are listed below. Consumers can define additional proprietary optional header fields. The six standard ones are:

1. EventReliability (value type **short**)
2. ConnectionReliability (value type **short**)
3. Priority (value type **short**)
4. StartTime (value type **TimeBase::UtcT**)
5. StopTime (value type **TimeBase::UtcT**)
6. Timeout (value type **TimeBase::UtcT**)

The Event Header is used for defining events and for filtering which events a consumer will receive on a per-message basis. In practice, much of the filtering is done at a higher level, which will be discussed later. The Event Body contains the actual content of the event instance. It too is divided into two parts, the filterable part and the remainder part. The filterable part is intended to contain the "most interesting fields" of the event, i.e., the ones that a consumer is most likely to base filtering decisions upon. The filterable part of the event body is also a sequence of name-value pairs with the name being a string and the value of type Any. The "fd_" prefix stands for "filterable data."

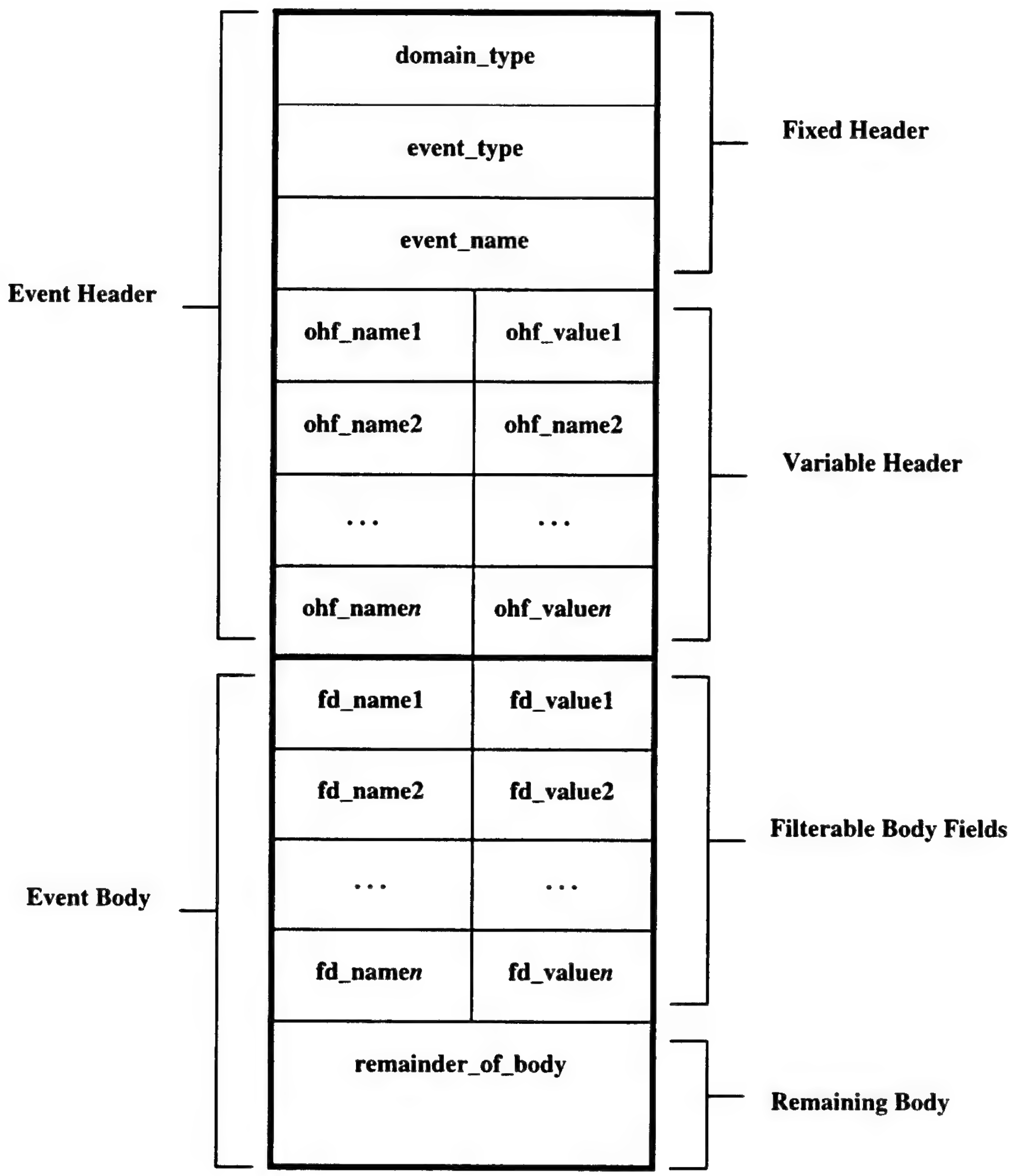


Figure A5. Format of a Structured Event

The intent is that different vertical domains will define standard mappings of specific event types into Structured Events. Such mappings will standardize the name-value pairs that make up the filterable part of a particular type of structured event. The proposed specification does, however, include a set of standard mappings for event types commonly used in the telecommunications domain. Backers of the specification presume that other vertical domains (e.g., healthcare, finance, transportation) will follow suit.

The last part of the Structured Event Body is a part called the Remaining Body which is defined to be of type Any. It provides a place to transmit any event data in addition to that, which is viewed as being interesting fields for which consumers are likely to define filters.

The backers of the proposed specification anticipate that implementors will be likely to also implement a structured event type repository that completely describes the structure of each structured event. The filterable name-value pairs are a type of meta-data about structured events. So such a repository would allow consumers to discover new instances of structured events that have been dynamically added to the system and construct filters for subscribing to them.

In addition to Structured Events there are *sequences* of structured events. Consumers can opt to get their structured event notifications in batches as opposed to one at a time. Furthermore, they can specify QoS properties like the maximum number of structured events in a batch. This capability is provided by the No. 4 variations of the ProxyPushConsumer and ProxyPushSupplier interfaces, which are named NCA::ProxyPushConsumer and NCA::ProxyPushSupplier respectively. We will not take up space in this appendix explaining this set of capabilities other than to remark that such a capability may have very relevant meaning for an ABIS style system because it would be a natural way to distribute real-time track information for maintaining a common operational picture such as envisioned as in the Single Integrated Air Picture.

There are also four different variations of each of the two basic data-pull Proxy objects. The variations of the ProxyPullConsumer objects are:

1. ProxyPullConsumer object
2. NCA::ProxyPullConsumer object
3. NCA::StructuredProxyPullConsumer object
4. NCA::SequenceProxyPullConsumer object

Similarly the ProxyPullSupplier object has the four different variations which are:

1. ProxyPullSupplier object
2. NCA::ProxyPullSupplier object
3. NCA::StructuredProxyPullSupplier object

4. NCA::SequenceProxyPullSupplier object

This gives us the total of 16 different types of Proxy objects shown in Figure A4 and completes our explanation of them. It is assumed that the reader is able to visualize how the data-pull Proxy objects function from our explanation of how the data-push Proxy objects work. It remains to explain the other objects in Figure A4.

A.2.3 Admin and Filtering Objects

Event Channel Proxy objects are created by means of the Admin objects. This policy is retained in the Notification Service. According to the proposed Notification Service specification on page 24, "Each Admin interface is essentially a factory that creates the Proxy interfaces to which clients will ultimately connect." Each Admin object "manages" the group of Proxy objects it has created. Admin objects are allowed to have QoS properties and filter objects themselves. The QoS properties are assigned to the Proxy objects they create. Consumers can later tailor these properties on a per-Proxy basis. But the filter objects associated with an Admin object are treated as a single unit, which applies at all times to all the Proxy objects created by the Admin object. Additional filter objects can be associated with a Proxy object on a per-Proxy basis, but the set of filter objects associated with the Admin object can only be modified by invoking Admin object interfaces.

The sharing of a set of filter objects among all Proxy objects created by a single Admin object is a powerful mechanism for creating a set of event subscriptions that can be shared by a group of consumers. Also, the filtering of a single event on behalf of a group of consumers is an optimizing feature since the filtering of a single event can be performed only once for a group of consumers. Supporting multiple Admin objects in a given notification channel enables the logical grouping of Proxy objects associated with the channel according to common subscription information. This feature enables the channel to optimize the servicing of a group of consumers interested in subscribing to the same set of events.

Notification Service style Proxy objects can have two different types of filters associated with them called *forwarding filters* and *mapping filters*. Forwarding filters can be associated with any Proxy object while mapping filters can only be associated with supplier type Proxy objects and affect the priority or lifetime properties of each event received by a supplier Proxy object. In order to avoid making this report unwieldy, we will not discuss the details of how these filters work or the constraint grammars they conform to. For our purposes it is enough to know that the consumers can specify their subscriptions in as much detail as they might desire.

There is another constraint grammar that the authors of this joint proposal considered that is already a CORBA specification. This is the Trader Constraint Language of the CORBA Trader Service. It was found to have too many deficiencies and ambiguities to be used, as is, for specifying data (event) elements in subscriptions. In order not to deviate any more than necessary from existing CORBA standards, the proposers are proposing to extend and modify the Trader Constraint Language for the default constraint grammar of the Notification Service.

This completes our discussion of the elements of Figure A4, which shows the structure of a Notification Channel. There are some additional explanations about notification channels that are relevant, which we now discuss.

A.2.4 Quality of Service Capabilities

The existing CORBA Event Service intentionally leaves the issue of Quality of Service as an implementation choice. Section 4.1.6 of the Event Service specification states: "Note that the interfaces defined in this chapter are incomplete for implementations that support strict notions of atomicity. That is, additional interfaces are needed by an implementation to guarantee that either all consumers receive an event or none of the consumers receive an event; and that all events are received in the same order by all consumers." Evidently, the authors of the CORBA Event Service had something like the ISIS or HORACE middleware in mind when they crafted this statement. These projects, by Ken Birman at Cornell, have pioneered the above-described services that the CORBA Event Service specifically states are not provided by the Event Service. The proposed Notification Service specification refers to the above quotation to show that the Event Service intentionally omits QoS considerations. But the Notification Service, while providing significant QoS capabilities, does not provide the capabilities ruled out by this Event Service quotation.

We next describe the QoS capabilities that the Notification Service does provide. The proposed Notification Service specification defines standard interfaces for controlling the QoS characteristics of event delivery. They claim to provide a QoS model but it is not shown in the specification. They state that it has four components and list them. They are:

1. A QoS property representation.
2. Accessor operations for setting and getting QoS properties at various levels of scope throughout an application.
3. QoS properties for notification.
4. A method for negotiating QoS and conflict resolution.

These "components" are discussed briefly in the specification. The first component, the QoS property representation turns out to be an adoption of the Trading Service representation method of representing properties as *(String, Any)* pairs. The idea here is to define a few QoS parameters that will be part of the standard but allow implementors the ability to extend this core set of parameters as they either gain knowledge of what QoS parameters are relevant with respect to this standard or as various vertically integrated domains require distinct types of QoS parameters. The Trading Service representation is such an expandable representation. Implementations of the proposed standard are required to be able to recognize all of the core QoS properties but are not required to implement them all. For those that they do not implement they are required to raise the NO_IMPLEMENT exception.

The next QoS model component is the accessor operations for setting and getting QoS properties. Accessor operations (*set_QoS* and *get_QoS*) are available at the following levels:

1. For a Notification Channel.
2. For an Admin Object associated with a Notification Channel.
3. For an individual Proxy object associated with an Admin object.

In addition, for Structured Events, QoS properties can be set on a per-message basis in the optional header fields of the message. These levels of scope form a hierarchy that reflects the ability to override QoS parameters at another level. QoS parameters set at the Notification Channel level establishes the default QoS parameters for message delivery for all groups, proxies and messages. Setting QoS parameters at the Admin object level overrides the Notification Channel level QoS parameters for all proxy objects that are members of the group of proxy objects that are managed by the Admin object. Setting QoS parameters at the individual proxy level overrides the group Admin or Notification Channel level default settings for a particular consumer object. Finally, setting the QoS parameters in an individual message overrides any other QoS parameters for that particular message.

The proposed specification points out that it may not always make sense to allow QoS properties to be overridden at all levels. The example that is given in the proposed specification is the setting of the reliable delivery parameter at the message level when the channel level has only been set to *best-effort*. It is not clear from the proposed specification whether the proposers are in favor of leaving how to deal with this situation to the implementors or if they intend for such overrides to be ignored, or what. In any event, there are some QoS issues here that seem to be left unresolved in the proposed specification.

Another QoS issue that the proposed specification points out deals with end-to-end QoS specification. In this case it is specifically stated that *it is the user's responsibility to deal with the problem of assuring that the QoS parameters are consistent across a whole communication path.* The problem is essentially this: when suppliers and consumers are connected via a channel such as specified in the proposed specification, there are three components of a message transversal between a supplier and a consumer. The first is from the supplier to the channel, the second is within the channel and the third is between the channel and the consumer. In this case, there is no direct communication between the supplier and the consumer so it is not possible to set the QoS parameters in one place that determines the QoS for the entire message transversal. This introduces the problem of *consistent QoS parameters across a path.* For example, the supplier component and the channel component both might have their QoS parameters set to *reliable delivery* but the consumer component may be set to *best-effort.* Without cooperation among the three components it is impossible to guarantee a specific QoS property.

As the reader may appreciate, there are other ways of handling these QoS issues that may seem more natural to the problem itself that are precluded by the retention of the concept of an Event Channel as a means of decoupling the suppliers and consumers. These other approaches may require less of the user of the service. The point here is not to compare these alternate approaches but to point out that these QoS consistency issues *arise from the underlying model of the (Notification) Event Channel.* In order to use these new proposed services effectively, one will have to have a clear understanding of what this model means to their particular application. The model has two positive features that the authors felt justified its acceptance. First, it is based on the same request-reply semantics as all other CORBA services (data being pushed as part of a request as well as requested) and therefore can be implemented largely from CORBA capabilities that already exist. Second, it is compatible with the existing CORBA Event Service. This model may not be natural for existing non-CORBA applications that are being converted to the CORBA environment. Users of the new proposed specification will need to take special care in specifying QoS and filtering parameters in this hierarchical connection model in order to obtain meaningful results.

The next component in the QoS model is the predefined Notification QoS properties. They all have names and are listed below by name.

1. Reliability
2. Priority
3. Expiry Time
4. Earliest Delivery Time

5. Order Policy
6. Discard Policy
7. Maximum Batch Size
8. Pacing Interval

Reliability Parameter. The Notification Service treats the reliability of specific events and the reliability of the connections that provide the transport for events between suppliers and consumers as separate issues. It defines two reliability subparameters to represent these two separate issues. These are the *EventReliability* subparameter and the *ConnectionReliability* subparameters. Each of these subparameters has two possible values, namely, *BestEffort* and *Persistent*. When both EventReliability and ConnectionReliability are set to BestEffort, no specific delivery guarantees are made.

When EventReliability is set to BestEffort and ConnectionReliability is set to Persistent the notification channel maintains all information about the consumers that are connected to it persistently so that connections will not be lost upon failure of the process that is executing the notification channel. Consumers connected to the channel using persistent object references may fail, but unless these object references raise an OBJECT_NOT_EXIST exception the channel will retry using them. Consumers that re-instantiate objects with these references will reconnect to their associated proxies. The channel will not however, have persistently stored any buffered messages at the time of failure. In summary, this combination of reliability parameters provides the capability to automatically reconnect consumers upon restart from a failure but no attempt will be made to save the messages that were buffered or were in transit at the time of failure.

The combination of having the EventReliability subparameter set to Persistent and the ConnectionReliability subparameters set to BestEffort is undefined in the proposed specification and is stated to have no meaning.

The combination of having both reliability subparameters set to Persistent provides the guarantee to deliver each event to all consumers registered to receive it at the time the event was delivered to the channel, within expiry limits.

Priority Parameter. The proposed Notification Service supports a large number of priorities (from -32767 to 32767) and the notification channel delivers messages in priority order. The default parameter setting for all events is 0. It is possible for consumers to override the priority assigned to a message through the use of mapping filters.

Expiry Time parameter. It is often desirable to indicate a time range in which an event is valid. This is accommodated by the expiry time parameter. If an event is not delivered within a specific time defined by the expiry time, then it is deleted. There are two ways of specifying expiry times. One is by specifying the *StopTime* property and the other is by specifying the *Timeout* property.

The StopTime property states an absolute expiry time (e.g., January 1, 2000) after which the event can be discarded. The Timeout property states a relative expiry time (e.g., 10 minutes from now) after which the event can be discarded.

It is possible for the consumer to override a Timeout property through the use of a mapping filter. The StopTime property can only be used on a per-message basis and thus is only supplied within the header of a structured event.

Earliest Delivery Time parameter. It is often desirable to indicate a time before which an event may not be delivered (e.g., June 1, 1999). This capability is accommodated via the Earliest Delivery Time parameter. This parameter has only one property, namely, the *StartTime* property. The StartTime property can only be used on a per-message basis and thus is only supplied within the header of a structured event.

Order Policy parameter. This QoS parameter is used to set the policy a given proxy uses to order the events it has buffered for delivery (either to another proxy or to a consumer). There are four predefined values that represent four different policies. They are:

1. *AnyOrder* – any ordering is permitted.
2. *FifoOrder* – Events are to be delivered in the order of arrival.
3. *PriorityOrder* – Events should be buffered in priority order so that higher priority events will be delivered before lower priority events.
4. *DeadlineOrder* – Events should be buffered in the order of shortest expiry deadline first so that events that are destined to timeout soonest should be delivered first.

This QoS parameter has no meaning if set on a per-message basis.

Discard Policy parameter. This QoS parameter allows the user of the Notification Service to specify the order in which the channel should begin discarding events in the case of an internal buffer overflow. There are five predefined values for this parameter corresponding to five different discard policies. They are:

1. *AnyOrder* – Discard any event on overflow.
2. *FifoOrder* – Discard the first event received first.
3. *LifoOrder* – Discard the last event received first.

4. *PriorityOrder* – Discard events in priority order starting with the lowest priority first.

5. *DeadlineOrder* – Discard events in the order of shortest expiry deadline first.

This QoS parameter has no meaning if set on a per-message basis.

Maximum Batch Size parameter. This QoS parameter is used to specify the maximum number of structured events that can be received by a consumer that receives sequences of structured events.

Pacing Interval parameter. This QoS parameter is used to specify the maximum time a notification channel will collect structured events into a sequence of structured events for a consumer before delivering the sequence. If the number of events received within a given *PacingInterval* equals or exceeds the *MaximumBatchSize* parameter, the consumer will receive a sequence of events whose length equals *MaximumBatchSize*. Otherwise the consumer will receive however many events arrived at the proxy supplier during the *PacingInterval*.

There are a number of other issues related to QoS properties such as negotiating QoS properties and the resolution of QoS conflicts that are contained in the Notification Service Specification. These are of a more technical nature and are therefore not included in this appendix. The reader is referred to the Notification Service (proposed) Specification for the full details of QoS properties and QoS issues.

ACRONYMS LIST

ABIS	Advanced Battlespace Information System
AITS	Advanced Information Technology Services
BODTF	Business Object Domain Task Force
CDF	Component Definition Facility
CDL	Component Definition Language
CORBA	Common Object Request Broker Architecture
C2	Command and Control
DARPA	Defense Advanced Research Projects Agency
DCE	Distributed Computing Environment
DDJPO	DARPA-DISA Joint Program Office
DISA	Defense Information Systems Agency
DSTC	Distributed Systems Technology Centre
GCCS	Global Command and Control System
IBM	International Business Machines Corporation
ICL	International Computers Limited
IDA	Institute for Defense Analyses
IIOP	Internet Interoperability ORB Protocol
ISO	Information Systems Office
OMA	Object Management Architecture
OMG	Object Management Group
OMG-STRUDL	OMG Structural Definition Language
ORB	Object Request Broker
QoS	Quality of Service
RFP	Request for Proposal
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SIAP	Single Integrated Air Picture
TCP/IP	Transmission Control Protocol/Internet Protocol
TIBNet	The Information Bus Network
TINA-C	Telecommunications Information Networking Architecture Consortium
TINA-ODL	TINA-C Object Definition Language

REFERENCES

General References

[ABIS TFR] *ABIS Task Force Report*, May 1996.

[CUS 97] Sally Cusack, *Rethinking Data Delivery*, Application Development Trends, July 1997.

[IDA P-3327] Edward A. Feustel, *Design Guidance for Converting Legacy Applications to Distributed Applications Using CORBA*, IDA Paper P-3327, February, 1997.

[MITRE 96] T.T. Bean, et. al., *Theater Air Defense Architecture for Joint Force Operations*, Working Note WN 96W127, September, 1996.

[OMG EMS] Event Management Services, Chapter 4 of the OMG Common Object Services Specification, 1995.

[WUSL 96a] Aniruddha Gokhale and Douglas C. Schmidt, *Measuring the Performance of Communication Middleware on High Speed Networks*, Proc. ACM SIGCOMM, August 1996.

[WUSL 96b] Douglas C. Schmidt et al., *Experience Developing an Object-Oriented Framework for High-Performance Electronic Medical Imaging using CORBA and C++,* Proc. Software Technology Applied to Imaging mini-conference, SPIE, January, 1996.

[WUSL 95] Douglas C. Schmidt et al., *Object-Oriented Components for High-speed Network Programming*, Proc. USENIX Conf. on Object-Oriented Technologies, June 1995.

OMG Approved Technologies

[AMS 98] Bea Systems, Inc. et. al. 1998. Messaging Service: Joint Revised Submission. ftp://ftp.omg.org/pub/docs/orbos/98-03-11.pdf. March 1998.

[AVS 98] IONA Technologies, Lucent Technologies, and Siemens-Nixdorf, 1997. Control and Management of Audio/Video Streams: Revised Submission. ftp://ftp.omg.org/pub/docs/formal/98-06-05.pdf. June 1998.

[EMS 97] Event Service Specification. 1997. Object Management Group, Framingham, MA. ftp://ftp.omg.org/pub/docs/formal/97-12-11.pdf. December 1997.

[MAS 97] GMD FOKUS and International Business Machines Corporation, 1997. Mobile Agent System Interoperability Facilities Specification: Update of Revised MASIF Submission. ftp://ftp.omg.org/pub/docs/orbos/97-10-05.pdf. October 1997.

[NOT 98] Bea Systems, Inc. et. al. 1998. Notification Service: Joint Revised Submission. ftp://ftp.omg.org/pub/docs/telecom/98-01-01.pdf. January 1998.

[OBV 98] Bea Systems, Inc. et. al. 1998. Objects by Value: Joint Revised Submission - with Errata. ftp://ftp.omg.org/pub/docs/orbos/98-01-18.pdf. January 1998.

RFP Submissions to OMG

[AAR 98] Alcatel Alsthom Recherche et. al. 1998. Realtime CORBA: Initial Submission. ftp://ftp.omg.org/pub/docs/orbos/98-01-08.pdf. January 1998.

[DA 97] Data Access. Multiple Interfaces and Composition: Revised Submission. ftp://ftp.omg.org/pub/docs/orbos/97-11-03.pdf. November 1997.

[DTSC 97] Distributed Systems Technology Centre (DSTC), International Computers Limited (ICL) and IONA Technologies, 1997. Multiple Interfaces and Composition: Revised Submission. ftp://ftp.omg.org/pub/docs/orbos/97-05-17.pdf. May 1997.

[GI 97] Genesis Development Corporation and Inline Software. Multiple Interfaces and Composition: Revised Submission. ftp://ftp.omg.org/pub/docs/orbos/97-12-35.pdf.

[HCVS 98] Highlander Communications and Visigenic Software, 1998. Realtime CORBA: Initial Submission. ftp://ftp.omg.org/pub/docs/orbos/98-01-14.pdf. January 1998.

[IBM 97] International Business Machines, 1997. Multiple Interfaces and Composition: Revised Submission. ftp://ftp.omg.org/pub/docs/orbos/97-05-23.pdf. May 97.

[IN 98] IONA Technologies and Nortel, 1998. Realtime CORBA: Initial Submission. ftp://ftp.omg.org/pub/docs/orbos/98-01-09.pdf. January 1998.

[LMFS 98] Lockheed Martin Federal Systems, 1998. Realtime CORBA: Initial Submission. ftp://ftp.omg.org/pub/docs/orbos/98-01-13.pdf. January 1998.

[OIS 98] Objective Interface Systems, 1998. Realtime CORBA: Initial Submission. ftp://ftp.omg.org/pub/docs/orbos/98-01-15.pdf. January 1998.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE May 1998	3. REPORT TYPE AND DATES COVERED Final

4. TITLE AND SUBTITLE Broker Performance	5. FUNDING NUMBERS DASW01-94-C-0054/ DASW01-97-C-0056 Task Order A-209
6. AUTHOR(S) Norman R. Howes, Edward A. Feustel	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Institute for Defense Analyses (IDA) 1801 N. Beauregard St. Alexandria, VA 22311-1772	8. PERFORMING ORGANIZATION REPORT NUMBER IDA Paper P-3439
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Deputy Director, Plans and Integration Information Systems Office Defense Advanced Projects Agency 3701 N. Fairfax Drive Arlington, VA 22203	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; unlimited distribution: 28 December 1998.	12b. DISTRIBUTION CODE 2A

13. ABSTRACT (Maximum 200 words)

The Common Object Request Broker Architecture (CORBA) defines a specification for developing brokers to mediate between distributed computing applications. Such brokers are being considered within the DARPA community as a means to effect data transfer among software applications supporting command and control (C2). The stressing timeliness demands for data transfer between distributed C2 nodes raises the question of how effective brokers are in satisfying such demands. To help address this question, this paper seeks to capture and synthesize what is known about broker performance and planned enhancements. Very limited quantitative performance data from controlled experiments could be found, and that found suggests current CORBA brokers might not meet the more stressing data transfer demands. Limited data on subscription brokers shows that they might offer better performance than current CORBA brokers. However, a number of planned new CORBA services were reviewed, and some – most notably the Notification and Data Stream Control Services – promise significantly enhanced data transfer performance. The paper concludes by recommending actions DARPA might take to better understand broker performance and to prepare for use of the new broker capabilities.

14. SUBJECT TERMS Common Object Request Broker Architecture (CORBA); Object Request Brokers; Subscription Brokers; Command and Control (C2).			15. NUMBER OF PAGES 88
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18